La Grammaire d'IEML (seulement le dictionnaire)

Pierre Lévy

La Grammaire d'IEML

Prof. Pierre Lévy, CRC, MSRC

9 novembre 2015

Table des matières

Liste des algorithmes

Table des figures

Liste des tableaux

Chapitre 1

Introduction à la grammaire d'IEML

1.1 Un ensemble de trois volumes

La grammaire d'IEML appartient à un ensemble de trois volumes qui explorent chacun à sa manière le même sujet, à savoir le nouvel univers de relations sémantiques calculables ouvert par le métalangage IEML [1].

- *La sphère sémantique* (*Computation, cognition, économie de l'information*) [2] contient les informations historiques, les références scientifiques et les justifications épistémologiques requises pour la pleine compréhension des motifs qui m'ont mené à l'invention du métalangage IEML. On y trouvera notamment une revue générale des travaux académiques recoupant les thèmes de ma recherche ainsi qu'une bibliographie extensive.
- *L'intelligence algorithmique* [3] expose une théorie scientifique originale de la cognition humaine. La théorie de la connaissance qui fonde l'intelligence algorithmique intègre – dès le niveau théorique – la dimension de circularité radicale (auto-référence, auto-organisation, auto-reproduction...) de la cognition humaine, ainsi que ses dimensions sémiotique, pragmatique, herméneutique, technique, sociale, incarnée et distribuée. IEML rend cette théorie techniquement opératoire. Cet ouvrage développe les conséquences concrètes qui résulteront de l'utilisation de l'intelligence algorithmique, qu'elles soient techniques, économiques ou sociales. Il donne enfin plusieurs exemples du contenu du dictionnaire d'IEML.
- Consacré à la *Grammaire d'IEML*, le présent volume possède un contenu essentiellement formel et technique. Il démontre notamment la calculabilité d'IEML et de sa sémantique, calculabilité qui fonde le projet de

1. IEML pour *Information Economy MetaLanguage*
2. Pierre Lévy, *La sphère sémantique. Computation, cognition, économie de l'information.* Hermès-Lavoisier, Paris et Londres, 2011, 410 p.
3. A paraître après 2014

l'intelligence algorithmique.

1.2 Finalités

1.2.1 Modéliser l'intelligence

Si l'on veut modéliser l'intelligence humaine, il est évident qu'il vaut mieux avoir en main une théorie scientifique de cette intelligence. Or l'intelligence humaine est intimement liée à la capacité de produire et de comprendre des expressions linguistiques et à la capacité de manipulation symbolique en général. C'est pourquoi on ne peut disposer d'un modèle scientifique de l'intelligence humaine sans avoir de modèle scientifique du langage, et notamment de la dimension sémantique du langage. J'ai donc construit (au moyen d'IEML) un modèle formel et calculable du fonctionnement du langage – modèle qui inclut aussi bien sa dimension *sémantique* que sa dimension *syntaxique*.

1.2.2 Coordonner la sémantique

L'unification des systèmes de transport et la mondialisation du commerce a poussé à l'unification des systèmes de poids et mesure et des systèmes de coordonnées géographique et temporel. A cause de l'interconnexion générale des informations et de la communication ubiquitaire, nous avons aujourd'hui besoin d'un système de coordonnées sémantiques universel tel que le métalangage IEML. Ce système de coordonnées sémantiques pourra...

— être utilisé comme langue-pivot entre les langues naturelles (IEML s'auto-traduit en langues naturelles),
— favoriser une mutation épistémologique des sciences humaines qui leur permettrait de tirer parti de l'abondance des données numériques,
— améliorer la gestion personnelle et collective des connaissances,
— fournir un miroir réflexif aux intelligences collectives qui s'auto-organisent dans le médium numérique.

1.3 Réponse à certaines critiques

1.3.1 IEML est possible... puisqu'il existe

Je n'ignore évidemment pas les puissants préjugés selon lesquels un langage universel, ou bien encore une sémantique calculable, sont « impossibles »... puisqu'elles n'existent pas encore. L'aviation, et plus encore les aéronefs plus lourds que l'air, ont été « impossibles » pendant des millénaires avant de devenir une réalité ordinaire. Le fait que les langues naturelles découpent chacune la réalité à leur manière ou que le sens dépende du contexte sont les deux évidences banales qui me sont le plus souvent opposées. Mais ce ne sont pas des objections valables. Premièrement, IEML permet justement de découper la réalité comme on le voudra. Deuxièmement, il permet de modéliser les contextes par des graphes

hypertextuels, des flux intensifs, des références multimédia précisément adressées et les règles formelles de jeux d'interprétation collective. Le « contexte » pourra donc être pris en compte dans les utilisations concrètes d'IEML.

1.3.2 IEML est souhaitable... parce qu'il augmente la liberté d'interprétation

Je voudrais répondre également par avance à ceux qui craignent que mon système ne limite la liberté des interprétations et la créativité conceptuelle. Un médium numérique équipé d'IEML abritera une mémoire herméneutique favorisant non seulement l'interopérabilité sémantique mais encore une puissance de différentiation conceptuelle et de diversification interprétative sans précédent dans l'histoire des technologies intellectuelles. Le métalangage n'impose aucune limite aux possibilités d'expression de nouveaux sens ou à l'ajout de significations indéfiniment plus précises. Il pourra pourra être librement utilisé par les conversations créatives en ligne qui s'organisent aujourd'hui sur les plates-formes de médias sociaux afin de modéliser réflexivement leur propre intelligence collective, de manière « perspectiviste ».

1.4 Organisation générale de la Grammaire d'IEML

La calculabilité d'IEML est la pierre angulaire de l'intelligence algorithmique. On dira qu'IEML est calculable à deux conditions : (1) si son système de signifiants répond à un groupe de transformations calculables et (2) si l'on peut transformer automatiquement ses signifiants en signifiés et *vice versa*. Or les signifiants d'IEML sont les expressions d'une algèbre commutative (un langage régulier fini) et ses signifiés sont des réseaux sémantiques (aussi complexes que l'on voudra) en langues naturelles. Puisque les signifiants d'IEML sont les expressions d'une algèbre commutative, alors la première condition est remplie. Examinons maintenant la seconde condition. Trois facteurs agissent de concert pour assurer la traduction automatique réciproque entre les signifiants et les signifiés d'IEML :

1) un codage de l'algèbre, le *Script*, qui établit une notation unique pour toutes les opérations équivalentes de l'algèbre,

2) une *grammaire* spécifiant toutes les étapes du calcul qui mène d'une expression en Script à un réseau sémantique en langue naturelle. Le calcul grammatical a évidemment besoin des données du dictionnaire pour arriver à ses fins. On trouvera dans ce livre la manière de calculer pas-à-pas des réseaux sémantiques en langue naturelle à partir de l'algèbre IEML.

3) un *dictionnaire* établissant une correspondance entre termes en IEML et termes en langues naturelles.

Le chapitre 2 ***Syntaxe*** présente d'abord le langage régulier sur lequel est fondé IEML, puis montre qu'il existe une transformation symétrique entre ce langage régulier et un graphe fractal hyper-complexe appelé le Rhizome IEML.

Le chapitre 3 suivant, ***Sémantique***, présente d'abord la structure du *dictionnaire* d'IEML : un ensemble de graphes paradigmatiques représentant les relations sémantiques entre les termes, puis la structure des *énoncés* en IEML : phrases, textes et hypertextes. Ces énoncés sont projetés sur la structure du Rhizome, ce qui permet de calculer leurs relations sémantiques internes et externes.

Le chapitre 4, ***Transcodage syntaxe-sémantique*** contient un des éléments-clés d'IEML puisqu'il formalise les algorithmes qui transforment la syntaxe d'IEML en sémantique. En d'autres termes, ce chapitre décompose pas à pas la transformation d'un objet mathématique - le langage régulier IEML - en objet linguistique : un ensemble de réseaux sémantiques exprimés en langues naturelles.

Finalement, le chapitre 5, ***Formalisation mathématique*** présente les aspects syntaxiques et sémantiques d'IEML sur un mode mathématique. J'y démontre notamment que le graphe hypercomplexe sous-jacent aux relations internes des réseaux sémantiques d'IEML (le Rhizome) est un groupe de transformations calculables (une « catégorie », au sens mathématique). La section 5.7 finale de ce dernier chapitre suggère un certain nombre de méthodes quantitatives (basées sur la théorie spectrale des graphes) qui pourraient être utiles pour la mesure des similarités structurales et des distances dans la sphère sémantique IEML (l'ensemble des circuits sémantiques générés par IEML).

Pour bien saisir le propos de la grammaire d'IEML, le lecteur est invité à se reporter fréquemment à la figure 1.1 que nous allons vais maintenant commenter. Les rectangles gris sont des *niveaux* et les espaces entre ces rectangles sont des *transcodeurs*.

1.5 L'Algèbre

Le niveau de base dans la partie la plus basse de la figure 1.1 représente une structure algébrique relativement simple que j'appelle l'Algèbre IEML ou, de manière abrégée, *l'Algèbre* (avec une initiale majuscule). Il s'agit d'un groupe de transformations calculables sur des variables construites à partir de l'alphabet {E, U, A, S, B, T} et dont les opérations additives et multiplicatives peuvent être en principe automatisées sans difficultés. Cette Algèbre est décrite de manière intuitive dans la section 2.1. Elle est formalisée au chapitre 5 dans la section 5.2 comme un langage régulier ; la section 5.3 examine sa structure de groupe et la section 5.4 prouve la calculabilité des opérations.

1.6 Le Script

Le Script IEML ou, de manière abrégée, le *Script* est représenté par le second niveau gris en partant du bas dans la figure 1.1. Il s'agit d'une notation particulière de l'Algèbre, notation qui est décrite dans la section 2.2. Les diverses expressions algébriques qui sont *égales* entre elles se traduisent par un

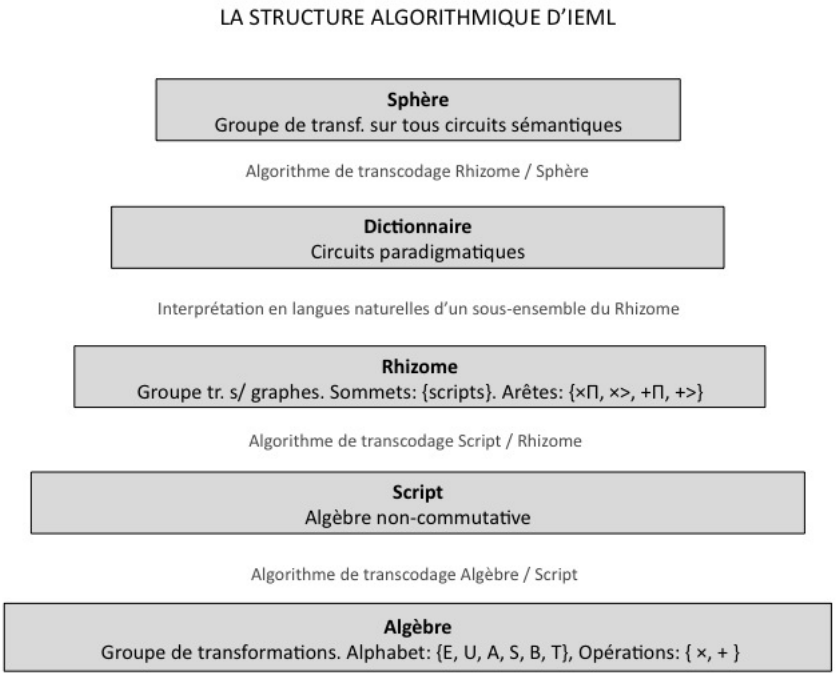

FIGURE 1.1 – La structure générale d'IEML

code unique en Script. Inversement, à tout code Script correspond une et une seule expression algébrique valide.

Entre le niveau de l'Algèbre et celui du Script opère un algorithme de transcodage réciproque qui traduit les expressions algébriques en Script et les codes Script en expressions algébriques. La structure de l'algorithme non trivial qui code l'Algèbre en Script est décrite de manière non-formelle à la section 2.3 et formalisée à la section 4.2.

1.7 Le Rhizome

La principale caractéristique du Script est de programmer la construction de la structure de graphes hyper-complexes qu'est le Rhizome. Le Rhizome est figuré par le troisième niveau gris de la figure 1.1 en partant du bas. Les sommets du Rhizome sont des codes scripts et ses arêtes représentent les relations opératoires entre les codes Script. Les quatre relations principales découlent des opérations algébriques : ordre ($\otimes >$) et symétrie ($\otimes \sqcap$) multiplicative, ordre ($\oplus >$) et symétrie ($\oplus \sqcap$) additive. La topologie du Rhizome est décrite à la section 2.4 et formalisée aux sections 5.5 et 5.6. *Cette topologie a été conçue afin de supporter les réseaux sémantiques (paradigmatiques, syntagmatiques, textuels et*

inter-textuels) qui organisent les unités de sens d'IEML, aussi bien « en langue » qu'« en parole ».

Entre le niveau du Script et celui du Rhizome opère un algorithme de transcodage qui traduit les codes scripts en rhizomes et les rhizomes en codes scripts. La structure de cet algorithme est décrite à la section 2.3 et formalisée à la section 4.3.

1.8 Le Dictionnaire

Dans le chapitre 2 sur la syntaxe, il ne sera question que d'algèbre, de codes, d'algorithmes et de graphes, sans aucun élément reconnaissable de linguistique ou de sémantique. Mais les trois premiers niveaux à partir du bas sur la figure 1.1 représentent la base indispensable d'une sémantique calculable. En effet, le Script établit une correspondance automatique entre l'algèbre IEML - d'une part - et les graphes hypercomplexes du Rhizome - d'autre part. C'est cette correspondance qui fait des graphes rhizomatiques les variables d'un groupe de transformations (plus exactement, une « catégorie » au sens mathématique) et qui autorise donc *le calcul sur les graphes hypercomplexes supportant les relations sémantiques entre les expressions du langage IEML.*

L'intervention humaine commence au troisième transcodeur inter-niveau à partir du bas sur la figure 1.1. A ce stade, des ingénieurs sémantiques choisissent des sous-ensembles du Rhizome et les interprètent en langues naturelles. Le résultat forme un dictionnaire, qui se présente comme un ensemble de circuits paradigmatiques. Les méthodes de construction et la structure des circuits paradigmatiques du dictionnaire sont décrits à la section 3.1.

1.9 Les circuits de la sphère sémantique

A partir des termes du dictionnaire et des règles de grammaire décrites à la section 3.2, il est enfin possible de construire et d'analyser automatiquement des circuits énonciatifs, c'est-à-dire syntagmatiques (propositions), textuels et hypertextuels. La section 3.3 esquisse la structure de l'algorithme capable de construire, à partir du Script et du Dictionnaire, l'ensemble des circuits hypercomplexes qui explicitent en langues naturelles les relations sémantiques entre les énoncés de la langue IEML. Cet algorithme est formalisé à la section 4.4.

1.10 Résumé

1. Il existe une correspondance entre les circuits sémantiques propres à la langue artificielle IEML et l'algèbre IEML qui est par définition calculable. *En d'autres termes, Il existe une correspondance automatique entre la syntaxe et la sémantique d'IEML.*

2. Les circuits paradigmatiques du dictionnaire établissent une correspondance automatique entre la langue artificielle IEML et les langues naturelles.

1.11 Références, index

Les références de la *bibliographie* se limitent aux aspects mathématiques. Pour tout ce qui concerne la linguistique et la sémantique, on consultera La sphère sémantique (2011). *L'index* final recense les principaux concepts des chapitres 2 et 3 (sur la syntaxe et la sémantique).

Chapitre 2

Syntaxe

« L'inconscient est structuré comme un langage » Lacan

Comme le lecteur le sait déjà, la reconnaissance, la production et la transformation des concepts discursifs en IEML est automatisable. Dans ce chapitre, nous allons aborder, de la manière la moins technique possible, le sous-bassement mathématique de la syntaxe d'IEML et deux de ses algorithmes de transcodage. Rappelons que les algorithmes seront formalisés au chapitre 4 et les mathématiques (ou tout au moins leurs fondements) dans le chapitre final (5) de cet ouvrage. Rappelons également que le chapitre 3, intitulé *Sémantique* traitera des aspects proprement linguistiques et sémantiques du métalangage, avec des exemples traduits en langue naturelle.

2.1 L'Algèbre

2.1.1 Les symboles initiaux

Les six symboles élémentaires {E, U, A, S, B, T} de notre Algèbre IEML répondent aux noms de vide (E), virtuel (U), actuel (A), signe (S), être (B) et chose (T).

2.1.2 Les séquences

2.1.2.1 Mode de construction des séquences

Considérons maintenant un algorithme capable de reconnaître et de manipuler les symboles élémentaires. Une de ses tâches consiste à fabriquer des séquences sémantiques en concaténant des symboles. On peut se représenter une séquence comme un objet fabriqué au moyen de symboles qui sont alignés l'un derrière l'autre, puis collés.

La longueur L d'une séquence est égale au nombre de symboles dont elle est composée. Ces séquences sont orientées, c'est-à-dire qu'elles ont un début et une fin, ou – si l'on veut – un symbole initial et un symbole terminal.

L0	L1	L2	L3	L4	L5	L6
1	3	9	27	81	243	729

TABLE 2.1 – Correspondance entre couches (Ln) et longueurs de séquences

2.1.2.2 Couches des séquences

Les séquences ne peuvent avoir que sept longueurs pré-définies : 1, 3, 9, 27, 81, 243, 729. On voit, premièrement, qu'un symbole peut être considéré comme une séquence de longueur L = 1 et que, deuxièmement, la longueur d'une séquence est toujours une puissance de 3 (3^0, 3^1, 3^2, 3^3, 3^4, 3^5, 3^6). On désignera une longueur de séquence, ou *couche*, par la puissance de 3 qui la définit. On parlera par exemple de séquence de *couche* L0, L1, L2, etc., jusqu'à L6 [1].

Pour construire une séquence de couche L0, on se contente de prendre un seul symbole. Pour construire une séquence de couche L1, on colle bout-à-bout trois séquences de longueur L0. Pour construire une séquence de couche L2, on colle bout-à-bout trois séquences de couche L1, et ainsi de suite.

En règle générale, pour construire une séquence de couche Ln (n étant supérieur à zéro et inférieur ou égal à 6), on colle bout-à-bout trois séquences de couche Ln–1.

2.1.2.3 Les trois sèmes d'une séquence : substance, attribut, mode

Les trois séquences de couche Ln–1 qui ont servi à construire la séquence de couche Ln sont toujours identifiables dans l'Algèbre. De plus, puisque les séquences ont un début et une fin, un algorithme peut déterminer laquelle des trois sous-séquences est la première, laquelle est la seconde et laquelle est la troisième. Ces trois sous-séquences sont appelées des sèmes. Le premier sème est appelé la substance de la séquence, le second est appelé l'attribut et le troisième le mode. Pour la signification grammaticale des sèmes, voir la section 3.2.3.

2.1.2.4 La multiplication

Les opérations algébriques pour construire des séquences sont appelées des *multiplications*. Le terme de « multiplication » est évidemment pris ici en un sens différent de celui de la multiplication entre des nombres. Ce sont ici des sèmes, c'est-à-dire des séquences sémantiques, qui sont les opérandes de la multiplication. Un algorithme peut aussi décomposer les séquences en trois sèmes de couche immédiatement inférieure. Un algorithme peut aussi transformer les séquences qu'il a construites, par exemple en remplaçant un sème par un autre.

1. Le L correspond à la première lettre de l'anglais *layer*.

2.1.3 Les ensembles de séquences

2.1.3.1 Types d'ensembles

Nous venons de voir que nos algorithmes étaient capable de construire, de reconnaître, de décomposer et de transformer des séquences. Nous allons voir maintenant qu'ils peuvent aussi emballer, reconnaître, déballer et transformer des *paquets* de séquences, des paquets de paquets, et ainsi de suite. Un paquet de séquences manipulé par un algorithme ne peut pas contenir de séquences identiques. De même, un paquet de sous-paquets ne peut contenir de sous-paquets identiques. Chaque séquence dans un sous-paquet et chaque sous-paquet dans un paquet est unique. C'est pourquoi nous appellerons ces paquets des *ensembles*. Les trois principales sortes d'ensembles manipulés par nos algorithmes du niveau algébrique sont :

— les *catégories*, qui sont des ensembles de séquences de même couche ;
— les *catsets*, qui sont des ensembles de catégories de même couche ;
— les *USL*, qui sont des ensembles de catégories de couches différentes, et qui peuvent donc toujours être organisés en ensembles de 1 à 7 catsets de couches distinctes.

Le terme d'*ensemble de séquences* est une expression très générale qui peut désigner une séquence, une catégorie, un catset, un USL, un ensemble d'USL, etc.

2.1.3.2 L'addition

Toutes les opérations de manipulations d'ensembles de séquences, telles que union, intersection ou différence symétrique sont appelées *opérations additives*, parce que le résultat de ces opérations peut toujours se représenter comme une union d'ensembles de séquences. Comme dans le cas de la multiplication sémantique, l'addition sémantique se prend évidemment en un sens différent de l'addition entre nombres.

2.1.4 Codage de l'Algèbre

Les opérations et les variables de notre système de séquences et d'ensembles de séquences sont intégralement définies et les ensembles de séquences sont en nombre fini. Les opérations sur les variables peuvent être décrites par des algorithmes. En d'autres termes, un circuit d'algorithmes du niveau de l'Algèbre est capable de construire, déconstruire et transformer des séquences ou des ensembles de séquences. De la même façon, un circuit d'algorithme peut décrire les séquences qu'il a construites et les ensembles de séquences qu'il a enveloppées.

On peut imaginer un très grand nombre de notations différentes de l'Algèbre IEML. Mais toutes ces notations sont fonctionnellement équivalentes du point de vue des résultats obtenus par les algorithmes qui peuvent décoder ces notations. La notation que adoptée couramment est fort proche de la « réalité » à décrire (le système des séquences). On marque les opérations multiplicatives par une pure et simple concaténation de symboles : par exemple {EUT} note une

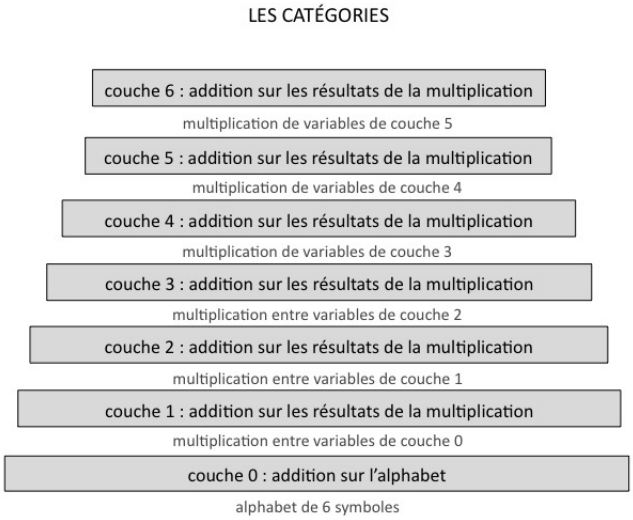

FIGURE 2.1 – Les couches de catégories en IEML

séquence de couche L1 dont E est la substance, U l'attribut et T le mode. On marque les opérations additives par une énumération utilisant la virgule. Par exemple {EUT, UAS, TUE} représente une catégorie de couche L1 contenant 3 séquences.

2.1.4.1 Caractérisation mathématique, calculabilité

A ce stade, IEML relève d'une algèbre commutative triviale : les multiplications cartésiennes en cascades qui génèrent ses séquences (les opérations multiplicatives) et les opérations qui assemblent des sous-ensembles de séquences (les opérations additives) peuvent être ramenées aux fonctions logiques les plus classiques. Vu sous l'angle des langages formels, l'Algèbre IEML relève des opérations bien répertoriées qui peuvent être accomplies sur les expressions d'un langage régulier fini. Du point de vue fonctionnel, c'est une catégorie (un groupe complexe) de transformations. On notera que le mot « catégorie » a ici deux sens distincts : (a) celui qu'il a en mathématiques (groupe de transformations complexe) et (b) celui qu'il a dans le vocabulaire technique d'IEML, à savoir un ensemble de séquences de même couche : voir la figure 2.1.

2.1.4.2 Il existe plusieurs notations pour un même résultat

J'insiste sur la commutativité opératoire de l'Algèbre IEML. Ses opérations additives sont évidemment commutatives.

Par exemple, le résultat de :

ISOMORPHIE ENTRE ALGÈBRE COMMUTATIVE ET TOPOLOGIE RHIZOMATIQUE

	ALGÈBRE COMMUTATIVE	TOPOLOGIE RHIZOMATIQUE
OPÉRATEURS + ×	Opérations sur ensembles de séquences	Filaments
VARIABLES	Ensembles de séquences	Bulbes
CODES	Ecriture algébrique	Ecriture Script

FIGURE 2.2 – Correspondance entre Algèbre et Rhizome

{{E, U, S}, {S, B, E}}
est le même que celui de :
{{E, S, B}, {E, U, S}},
à savoir :
{E, U, S, B}.

Notons aussi que l'associativité des opérations additives et multiplicatives permet d'arriver au même résultat de bien des manières.

Par exemple,
{{S{B, A}T}, {SET}}
donne le même résultat que :
{S{E, A, B}T},
à savoir :
{SET, SAT, SBT}.

Même en restant à l'intérieur des limites d'un seul système de notation algébrique, il existe donc plusieurs manières de coder le même ensemble de séquences.

2.2 Le Script

2.2.1 Convention d'écriture : Script et script

On écrira *Script* (avec un S majuscule) pour désigner **le code** de notre système de topologie rhizomatique et *script* (avec un s minuscule) pour désigner **une expression particulière** de ce code. La notation Script remplace la notation STAR dont il était question dans *La sphère sémantique* (2011).

2.2.2 Correspondance bi-univoque entre ensembles de séquences et scripts

Il existe plusieurs systèmes de notation possibles de l'Algèbre IEML. Le Script est une forme d'écriture particulière de l'Algèbre. Cela signifie qu'une notation en Script est *ipso facto* une notation de Algèbre IEML. Il en résulte qu'un script désigne un ensemble de séquences et un seul. Le transcodage du Script IEML vers l'Algèbre IEML est donc assuré. Ce transcodage ne pose pas de problème parce que le premier est un cas particulier du second. En revanche, le transcodage en sens inverse, de l'Algèbre au Script, est plus complexe. C'est pourquoi il existe un *algorithme de transcodage*[2] qui transforme les expressions de l'Algèbre en scripts. Cet algorithme respecte la contrainte selon laquelle il existe toujours un script et un seul pour décrire un ensemble de séquences. En somme :

— toutes les expressions différentes de l'Algèbre commutative qui désignent le même ensemble de séquences sont traduites par un script et un seul ;
— un script distinct désigne un ensemble de séquences et un seul et il a toujours une traduction en Algèbre commutative.

Cette correspondance bi-univoque entre les deux codes a d'importantes conséquences. En effet, les variables, les opérations et le code d'un système formel sont solidaires. Si l'on peut passer automatiquement du code d'un système formel A au code d'un système formel B et *vice versa*, alors on peut aussi passer automatiquement – par l'intermédiaire du code! – des variables et des opérations du système A aux variables et aux opérations du système B (et *vice versa*).

2.2.3 Correspondance bi-univoque entre scripts et rhizomes

Un script code non seulement un *ensemble de séquences*, mais il doit encore coder simultanément, et de manière non-ambiguë, un *graphe* complexe appelé rhizome (voir plus bas la section : 2.4). En d'autres termes :

— chaque script IEML distinct est interprété par un algorithme comme la description d'un rhizome distinct et un seul (voir plus bas la section 2.5 comment fonctionne cet algorithme) ;
— un rhizome distinct est décrit ou reconnu par un algorithme sous la forme d'un script distinct et un seul.

2. Voir plus bas les sections 2.3 et 4.2

2.2.4 Correspondance bi-univoque entre ensembles de séquences et rhizomes

Il existe une correspondance bi-univoque entre ensembles de séquences et scripts, d'une part, et entre scripts et rhizomes, d'autre part. Dès lors, il existe nécessairement une correspondance bi-univoque entre ensembles de séquences et rhizomes.

2.2.5 Solidarité entre les opérations du calcul algébrique et celles du calcul rhizomatique

Puisqu'il existe une correspondance bi-univoque entre ensembles de séquences et rhizomes, alors il existe aussi une correspondance entre le calcul algébrique qui construit, transforme et décrit les ensembles de séquences et le calcul rhizomatique qui construit, transforme et décrit les rhizomes.

2.2.6 Règles de notation du Script

2.2.6.1 Opérateurs

Multiplication : . - ' , _ ; Au lieu de se marquer par une simple concaténation de symboles, la multiplication se note en Script par des *signes de ponctuation* indiquant les différentes couches. Ces signes de multiplication facilitent l'analyse syntaxique automatique (*parsing*) des expressions et permettent de repérer les sèmes (c'est-à-dire les opérandes de la multiplication) visuellement, couche par couche. Les signes de multiplication en Script se présentent comme suit :

L0 **:**
L1 **.**
L2 **-**
L3 **'**
L4 **,**
L5 **_**
L6 **;**

Par exemple, la séquence :
STAUBSTAUESSUBSTAUEBSTBSAAU
se note en script :
S:T:A:.U:B:S:.T:A:U:.-E:S:S:.U:B:S:.T:A:U:.-E:B:S:.T:B:S:.A:A:U:.-'

Addition (+) Au lieu de se marquer par une énumération entre accolades, l'addition se note en Script par le signe + ainsi que par une parenthèse ouvrante et une parenthèse fermante.

Par exemple, {UAS, AUT, TUA} se note en Script (U:A:S:. + A:U:T:. + T:U:A:.)

Algèbre	Script	Signification
{U, A}	O:	verbe, processus
{S, B, T}	M:	nom, entité
{U, A, S, B, T}	F:	plénitude (absence de vide)
{E, U, A, S, B, T}	I:	information

TABLE 2.2 – Abréviations d'additions de couche L0

2.2.6.2 Délimitateurs

Délimitateurs de catégories Les catégories distinctes sont séparées par une barre oblique "/". Par exemple :

(U:A:S:. + A:U:T:. + T:U:A:.) / S:T:A:.U:B:S:.T:A:U:.-E:S:S:.U:B:S:.T:A:U:.-E:B:S:.T:B:S:.A:A:U:.-'

Délimitateurs d'USL * ** On se souviendra que les USL sont des ensembles de catégories distinctes. S'il est nécessaire de délimiter un USL en Script, on le fait précéder d'une étoile « * » et suivre de deux étoiles « ** ».

Par exemple :

*(U:A:S:. + A:U:T:. + T:U:A:.) / S:T:A:.U:B:S:.T:A:U:.-E:S:S:.U:B:S:.T:A:U:.-E:B:S:.T:B:S:.A:A:U:.-'**

2.2.6.3 Abréviations obligatoires

Abréviations d'additions de couche L0 L'abréviation des quatre additions de couche 0 figurant dans le tableau 2.2 est obligatoire.

Abréviation des multiplications de E: L'emploi de l'abréviation pour les multiplications de E: est obligatoire. Lorsqu'un E: est multiplié par d'autres E: jusqu'à la fin d'une catégorie, on se contente de marquer les couches sans les remplir de E:

J'utilise le signe # avant les expressions incorrectes en Script.

Exemple 1

On n'écrit pas #E:E:E:. mais *E:.**

Exemple 2

On n'écrit pas #O:E:E:.E:M:E:.E:E:E:.- mais *O:.E:M:.E:.-**

Exemple 3

On n'écrit pas #U:E:E:.E:E:E:.E:E:E:.- mais *U:.-**

Abréviation de 25 séquences de couche 1 L'emploi des 25 lettres minuscules codant certaines séquences de couche 1 est obligatoire.

2.2.7 Ordre standard

L'ordre standard du Script fournit des procédures régulières pour ranger les catégories, les catsets, et les USL.

	Les 25 lettres minuscules en Script (couche 1)				
Attribut → / Substance ↓	U	A	S	B	T
U	UUE wo. réfléchir interne	UAE wa. mouvoir	USE y. savoir	UBE o. vouloir	UTE e. pouvoir
A	AUE wu. percevoir	AAE we. réfléchir externe	ASE u. énoncer	ABE a. s'engager	ATE i. faire
S	SUE j. mutation de signifiant	SAE g. mutation documentaire	SSE s. pensée	SBE b. langage	STE t. mémoire
B	BUE h. mutation de signifié	BAE c. mutation personnelle	BSE k. société	BBE m. affect	BTE n. monde
T	TUE p. mutation de référent	TAE x. mutation matérielle	TSE d. vérité	TBE f. vie	TTE l. espace

FIGURE 2.3 – 25 lettres minuscules

2.2.7.1 Ordre de couche

On range toujours les ensembles de séquences par ordre de couche. Par exemple, on range les ensembles de couche 0, avant les ensembles de couche 1, les ensembles de couche 1 avant les ensembles de couche 2, etc.

2.2.7.2 Ordre de taille

L'ordre de taille correspond à l'ordre croissant du nombre de séquences singulières par catégorie, à l'ordre croissant du nombre de catégories séparées pour les catsets, etc. Il s'agit toujours de ranger les ensembles de séquences de même type par nombre croissant d'éléments.

Par exemple :

A:B:S:. ne contient qu'une seule séquence singulière {ABS}

O:B:S:. en contient deux {UBS, ABS}

M:B:S:. en contient trois {SBS, BBS, TBS}

E:	U:	A:	S:	B:	T:
1	2	4	8	16	32
2^0	2^1	2^2	2^3	2^4	2^5

TABLE 2.3 – Valeur numérique des primitives

O:M:S:. en contient six {USS, UBS, UTS, ASS, ABS, ATS}

2.2.7.3 Ordre alphabétique des primitives, des caractères et des lettres minuscules

Chaque symbole primitif possède une valeur numérique (voir le tableau 2.3). La valeur numérique d'une addition de primitives résulte de l'addition arithmétique des valeurs numériques des opérandes. Par exemple, la valeur numérique de (S:+B:) est 24 et la valeur numérique de O: est 6.

Chacune des 63 additions de couche L0 [3] est considérée comme un *caractère* du Script. L'ordre alphabétique du script range les caractères selon leur valeur numérique croissante.

Conformément aux principes qui viennent d'être énoncés, l'ordre alphanumérique des 10 lettres capitales est

E: U: A: O: S: B: T: M: F: I:

et celui des 25 lettres minuscules (voir la figure 2.3) est le suivant :

wo. wa. y. o. e. wu. we. u. a. i. j. g. s. b. t. h. c. k. m. n. p. x. d. f. l.

Pour comparer les lettres minuscules et capitales il faut traduire les minuscules en majuscules. Par exemple, pour comparer y. et E:S:E:. il faut utiliser U:S:E:. au lieu de y.

2.2.7.4 Ordre des catégories de même couche

L'ordre standard du Script range les catégories de même couche par taille croissante, puis par ordre alphabétique.

Exemple 1

Les catégories de couche 1 qui suivent sont rangées selon l'ordre des catégories. Les nombres en gras indiquent la **taille** des catégories qui suivent.

1
E:B:T:.
A:B:E:.
A:B:S:.
3
E:B:M:.
M:S:E:.
5
U:F:E:.
6

3. C'est-à-dire l'ensemble des sous-ensembles de {E, U, A, S, B, T} moins l'ensemble vide.

E:S:I:.
U:M:O:.
U:I:E:.
A:T:I:.
9
U:M:M:.
30
I:E:F:.
180
I:F:I:.
Exemple 2
Rangement par ordre alphabétique de 4 catégories de même taille de couche L2 :
O:O:.M:M:.-
O:M:.O:M:.-
O:M:.M:O:.-
M:M:.O:O:.-

2.2.7.5 Ordre des opérandes d'une addition

Les opérandes des additions sont toujours rangées par ordre standard.
Exemple 1
On n'écrit jamais #(S:+E:+U:) mais toujours *(E:+U:+S:)**
Exemple 2
On n'écrit jamais #(M:+U:) mais toujours *(U:+M:)**

2.2.7.6 Ordre des catsets

Les catsets sont en principe déjà rangés "en interne" selon l'ordre des catégories.

1. Un ensemble de catsets est d'abord rangé par couches
2. Les catsets de même couche sont rangés par ordre croissant de taille (c'est-à-dire par nombre de catégories).
3. Les catsets de même couche et de même taille sont rangés par ordre alphabétique de leur première catégorie. Si les premières catégories sont identiques, on les range par ordre alphabétique de leur seconde catégorie, et ainsi de suite jusqu'à ce qu'ils soient tous ordonnés.

2.2.7.7 Ordre des USL

Les USL sont en principe déjà rangés en catsets de couches distinctes (de 1 à 7 catsets), les catsets étant eux-même rangés par ordre de catégories. Les USL sont ordonnés par couche croissante, dans l'ordre de leurs catsets. C'est-à-dire que les USL commençant par un catset de couche 0 viendront d'abord, puis les USL commençant par un catset de couche 1, et ainsi de suite.

2.2.8 Contraintes d'écriture en Script***

Il est entendu que les scripts peuvent toujours être interprétés comme des instructions d'opérations additives et multiplicatives sur des séquences. Mais les scripts doivent aussi pouvoir être interprétés comme des instructions d'opérations « topologiques » de construction ou d'identification de rhizomes. C'est notamment pour répondre à cette seconde contrainte que le Script se présente comme une algèbre non-commutative et non-associative. Il n'existe qu'un script et un seul qui puisse coder un rhizome distinct.

2.2.8.1 Caractère non-commutatif et non-associatif de l'addition en Script

Non-commutativité Les opérandes des additions entre catégories de même couche sont rangées dans l'ordre standard. Cela implique l'absence de commutativité de l'addition.

Exemple
On *ne* peut *pas* écrire que :
*(U:+A:)** est identique à #(A:+U:)
puisqu'on ne peut pas écrire :
#(A:+U:)

Non-associativité Il est impossible d'écrire en Script *plusieurs* additions différentes entre catégories de couche Ln à l'intérieur d'un sème (opérande de multiplication) de couche Ln. On ne peut pas écrire non plus une addition d'addition.

Exemple
L'écriture :
#(y.+ a.)+(F:E:O:.+O:F:E:.)
est incorrecte en Script. Il faut écrire :
*(y. + a.+ O:F:E:.+F:E:O:.)**

2.2.8.2 Minimisation du nombre de multiplications en Script

On ne peut pas écrire en Script une addition de plusieurs multiplications de couche Ln quand on peut écrire le même résultat par une seule multiplication d'additions de couche Ln–1. En d'autres termes, on utilise toujours le plus petit nombre possible de multiplications de couche Ln. Cela implique une absence d'associativité dans la composition de l'union et de la multiplication en Script.

Exemple 1
On ne peut pas écrire :
#(U:S:A:. + U:S:B:. + U:T:A:. + U:T:B:.)
mais on doit écrire :
*U:(S:+T:)(A:+B:).**

On voit que l'expression incorrecte comprend quatre multiplications de couche L1 alors que l'expression correcte n'en comprend qu'une. L'expression correcte minimise le nombre de multiplications.

Exemple 2

Considérons la catégorie {USE, UBE, UTE, ASE, ABE, ATE} en Algèbre commutative.

Cette catégorie ne peut pas être représentée en Script par :

#((y.+ o. + e.) + (A:M:.))

ni par :

#(U:M:. + A:M:.)

La séquence ne peut être représentée en Script que par :

*O:M:.**

2.2.8.3 Choix entre plusieurs scripts offrant le même nombre minimal de multiplications

Lorsqu'il existe plusieurs scripts qui offrent le même nombre minimal de multiplications, on prend systématiquement celui qui est situé en premier dans l'ordre standard.

Exemple

Considérons la catégorie {USS, UTS, UTB, STB, SSB, SSS} notée en Algèbre commutative.

Il n'existe pas de scripts qui diminuent le nombre de multiplications au-delà de trois. D'autre part, il existe deux scripts qui offrent une traduction de l'expression algébrique avec seulement trois multiplications. Entre ces deux scripts, il faut donc choisir celui arrive le premier dans l'ordre standard. L'autre est incorrect.

*(U:(S:+T:)S:. + S:S:(S:+B:). + (U:+S:)T:B:.)**

#(U:T:(S:+B:). + S:(S:+T:)B:. + (U:+S:)S:S:.)

2.3 Principes de l'algorithme de transcodage de l'Algèbre vers le Script***

2.3.1 Architecture générale de l'algorithme de codage en Script pour une catégorie de couche Ln

L'opération de remplacement est expliquée à la section suivante.

1. Opération de remplacement (récursive) pour la catégorie de couche Ln
2. Opération de remplacement (récursive) pour les catégories de couche Ln–1
3. etc.
4. Opération de remplacement (récursive) pour les catégories de couche L1

2.3.2 Opération de remplacement sur une catégorie de couche Lk

Chaque fois que n (n le plus grand possible) séquences appartenant à la même catégorie IEML de couche L ont deux sèmes semblables (par exemple, la substance et le mode), elles sont remplacées par une seule séquence, c'est-à-dire par une seule multiplication. Ce remplacement se fait en utilisant les deux règles suivantes:

1. On cherche toujours à minimiser le nombre de multiplications résultant en une catégorie de couche Ln
2. Lorsque plusieurs catégories s'offrent pour obtenir le même nombre de multiplications minimal, on prend systématiquement la catégorie située en premier dans l'ordre standard décrit à la section 2.2.7.

2.3.3 Réitération de l'opération de remplacement

On répète l'opération de remplacement sur le résultat de la première opération, en boucle, jusqu'à ce que l'on ne trouve plus de séquences de caractères ayant deux sèmes semblables.

2.3.4 Exemples pour l'algorithme de codage en Script

Exemple 1 (couche L1) : {USE, UBE, UTE, ASE, ABE, ATE}

— USE, UBE, UTE ont 2 sèmes semblables, et donc peuvent être représentées par:
U:(S:+B:+T:)E:.
— ASE, ABE, ATE ont 2 sèmes semblables, et donc peuvent être représentées par:
A:(S:+B:+T:)E:.
— On obtient donc:
#U:(S:+B:+T:)E:.+A:(S:+B:+T:)E:.
— Les deux opérandes de l'union obtenue ont deux sèmes en commun. Donc on la représente ainsi:
#(U:+A:)(S:+B:+T:)E:.
— Afin d'obéir aux contraintes de notation abrégée, le dernier résultat est représenté ainsi:
*O:M:.**

Exemple 2 (couche L1) : {AUB, ABE, SUE, SBE, TUE, TBE}

— ABE, SBE et TBE peuvent être représentées par:
(A:+S:+T:)B:E:.
— SUE et TUE peuvent être représentées par
(S:+T:)U:E:.
— On obtient alors:
*(A:U:B:.+(S:+T:)U:E:.+(A:+S:+T:)B:E:.)**
Remarque : l'ordre standard oblige à ranger les opérandes de l'addition

par ordre de taille croissante. La substance du premier opérande est de taille 2 tandis que la substance du second opérande est de taille 3.

Exemple 3 (couche L2) : {AUBABESUE, AUBABEABE}

— {AUBABESUE, AUBABEABE} s'écrit *A:U:B:.A:B:E:.(A:B:E:.+S:U:E:.)-**

Exemple 4 (couche L2) : {USEUBEUTE, ASEABEATE, ASEABEUTE}

— Remplacement pour la couche 2
#(U:S:E:.U:B:E:.U:T:E:.- + A:S:E:.A:B:E:.(U:T:E:.+A:T:E:.)-)

— Remplacement pour la couche 1
*(U:S:E:.U:B:E:.U:T:E:.- + A:S:E:.A:B:E:.(U:+A:)T:E:.-)**

2.4 Le Rhizome***

2.4.1 Bulbes et filaments

Le terme de rhizome est ici employé dans un sens qui n'a que de lointains rapports avec celui qu'il possède en botanique et se réfère plutôt au sens philosophique de *graphe fractal régulier plus complexe qu'une structure d'arbre classique*[4]. Un rhizome IEML est fondamentalement un graphe dont les sommets, appelés *bulbes* sont étiquetés par un script et dont les arêtes, appelées *filaments*, sont étiquetées selon les relations qu'elles établissent entre les bulbes. Puisqu'il n'existe que quatre types de relations entre les bulbes, il n'y a que quatre étiquettes de filaments[5]. Les relations entre les bulbes découlent directement des deux opérations (addition et multiplication) et se nomment ainsi :

1. Ordre additif
2. Symétrie additive
3. Ordre multiplicatif
4. Symétrie multiplicative

Du point de vue de la théorie des graphes, les relations d'ordre construisent des arbres tandis que les relations symétriques construisent des cliques.

Dans les arbres additifs, le nombre de feuilles que peut avoir une racine n'est pas spécifié. Les feuilles et les racines des arbres additifs sont des bulbes de même couche.

Dans les arbres multiplicatifs, chaque racine ne peut avoir que trois feuilles, correspondant aux trois opérandes de la multiplication. Les feuilles des arbres multiplicatifs appartiennent toujours à la couche immédiatement inférieure à la couche de leurs racines.

4. Voir Deleuze et Guattari, *Mille plateaux*, Minuit, Paris, 1980, particulièrement l'introduction.

5. Nous verrons à la section **??** qu'il en existe 4 de plus, donc 8 en tout, mais nous pouvons négliger cela pour le moment.

FILAMENTS	**Multiplication** $\otimes$	**Addition** $\oplus$
Ordre $>$	$\otimes >$	$\oplus >$
Symétrie $\sqcap$	$\otimes\sqcap$	$\oplus\sqcap$

TABLE 2.4 – Les 4 types de filaments d'un rhizome IEML

2.4.2 Notation des rhizomes et définition des quatre types de filaments

Du fait des contraintes spéciales qui pèsent sur le Script (établir une relation bi-univoque entre l'Algèbre et le Rhizome), la *description détaillée des rhizomes ne peut se faire en Script.* Nous sommes donc obligés de faire appel à une notation « topologique » spéciale, distincte de celle de l'Algèbre et du Script. En principe, cette notation n'est utilisée que pour décrire la structure du Rhizome.

2.4.2.1 Bulbes

Un bulbe (c'est-à-dire un sommet de graphe rhizomatique) est ici noté par un petit cercle avant le script qui l'étiquette (sans étoiles).

Par exemple : °M:S:U:.

2.4.2.2 Filaments d'ordre additif

La relation d'ordre additive se note par le signe $\oplus >$

Le signe d'ordre additif indique la relation entre le résultat (à gauche du signe) et l'opérande (à droite du signe) d'une addition. Rappelons que, à l'intérieur d'une catégorie, les bulbes en relation d'ordre additif appartiennent toujours à la même couche.

Par exemple, °M:S:U:. $\oplus >$ °B:S:U:. indique que °M:S:U:. *contient* additivement le bulbe °B:S:U:. c'est-à-dire que l'ensemble de séquences étiquetant le bulbe °B:S:U:. *est un sous-ensemble de* l'ensemble de séquences étiquetant le bulbe °M:S:U:.

2.4.2.3 Filaments de symétrie additive

La relation additive symétrique se note par le signe $\oplus\sqcap$ entre crochets.

Le signe de symétrie multiplicative indique que les bulbes qu'il connecte sont des opérandes de la même addition. Il existe invariablement un *graphe complet* de relations symétriques additives entre les opérandes d'une addition et ces opérandes, bien entendu, appartiennent toujours à la même couche (à l'intérieur d'une catégorie).

Par exemple, [°S:S:U:. $\oplus\sqcap$ °B:S:U:. $\oplus\sqcap$ °T:S:U:.] indique que chacun des trois bulbes est en relation de symétrie additive avec les deux autres, ou que °S:S:U:. °B:S:U:. et °T:S:U:. sont les opérandes d'une même opération d'addition.

2.4.2.4 Notation compacte des relations additives correspondant à une seule addition

Au lieu d'indiquer successivement plusieurs relations d'ordre comme dans :
°M:S:U:. $\oplus >$ °S:S:U:.
°M:S:U:. $\oplus >$ °B:S:U:.
°M:S:U:. $\oplus >$ °T:S:U:.
On peut écrire l'ensemble des relations d'ordre et de symétrie additive correspondant à une seule addition de la manière suivante :
°M:S:U:. $\oplus >$ [°S:S:U:. $\oplus\sqcap$ °B:S:U:. $\oplus\sqcap$ °T:S:U:.]

2.4.2.5 Filaments d'ordre multiplicatif

Le filament d'ordre multiplicatif se note par le signe $\otimes >$
Le signe d'ordre multiplicatif indique la relation entre le résultat (à gauche du signe) et l'opérande (à droite du signe) d'une multiplication. Rappelons que l'opérande appartient toujours à la couche immédiatement inférieure à celle du résultat.
Par exemple :
°M:S:U:. $\otimes >$ substance °M:
°M:S:U:. $\otimes >$ attribut °S:
°M:S:U:. $\otimes >$ mode °U:
Au lieu d'indiquer successivement les trois relations d'ordre multiplicatif, on peut utiliser la notation suivante, dans laquelle l'ordre des sèmes indique leurs rôles distincts :
°M:S:U:. $\otimes >$ [°M: $\otimes$ °S: $\otimes$ °U:]

2.4.2.6 Filaments de symétrie multiplicative

Le filament de symétrie multiplicative se note par le signe $\otimes\sqcap$
Le signe de symétrie multiplicative indique que les deux bulbes de couche n qu'il connecte ont deux de leurs sèmes de couche n-1 en rôle inversé tandis que le troisième sème de couche n-1 est identique. Les bulbes en relation de symétrie multiplicative appartiennent nécessairement à la même couche.
Par exemple : °M:S:U:. $\otimes\sqcap$ °S:M:U:.

2.5 Principes de l'algorithme de transcodage du Script vers le Rhizome***

Un script est considéré comme un programme de construction de rhizome. La lecture automatique du script se ramène ultimement à des opérations d'identification des bulbes et de tissage de filaments entre les bulbes identifiés, c'est-à-dire à la construction d'un rhizome. Le mot « caractère » utilisé plus bas se réfère aux 63 sous-ensembles de {E, U, A, S, B, T} moins l'ensemble vide. Il ne s'agit donc pas des seulement des dix symboles {E, U, A, O, S, B, T, M, F, I}.

2.5.1 Structure générale de l'algorithme de construction d'un rhizome à partir d'un script de couche Ln

La construction de rhizome à partir d'un script de couche Ln (n entre 0 et 6) se décompose ainsi :

1. Construction pour la couche Ln,
2. Construction pour la couche Ln–1,
3. ...
4. Construction pour la couche L0

2.5.2 Structure générale de la routine de construction de rhizome pour une couche

1. Premier tour
   (a) Identification des bulbes de couche Lk en relation d'addition livrés par le script.
   (b) Tissage des filaments d'ordre et de symétrie additive de couche Lk entre les bulbes identifiés en (a).
   (c) Tissage des filaments d'ordre multiplicatif (racines à la couche Lk–1) des bulbes identifiés en (a).
2. Second tour : construction successive caractère par caractère des bulbes de couche Lk identifiés au premier tour.
3. Troisième tour : construction successive caractère par caractère des bulbes de couche Lk identifiés au second tour.
4. ...
5. Avant dernier tour : on continue ainsi jusqu'à ce que tous les bulbes identifiés correspondent à des séquences singulières.
6. Dernier tour : on tisse les filaments de symétrie multiplicative entre les bulbes identifiés aux tours précédents.

2.5.3 Passage d'une couche à la couche inférieure

La construction pour chaque couche livre les racines multiplicatives de couche immédiatement inférieure comme on le voit au point 1 (c) de la section 2.5.2. *Ce sont ces racines qui servent de base à la construction du rhizome à la couche inférieure, et ainsi de suite récursivement.*

2.5.4 Structure générale de la routine de construction de bulbes

1. Construction à partir du premier caractère

   (a) Identification des caractères en relation additive *contenus* dans le premier caractère, identification des bulbes correspondants.
   (b) Tissage des filaments d'ordre et de symétrie additive de couche Lk entre les bulbes identifiés en (a).
   (c) Tissage des filaments d'ordre multiplicatif (racines à la couche Lk–1) des bulbes identifiés en (a).

2. Construction à partir du second caractère
   (a) Identification des caractères en relation additive *contenus* dans le second caractère, identification des bulbes correspondants.
   (b) Tissage des filaments d'ordre et de symétrie additive de couche Lk entre les bulbes identifiés en (a).
   (c) Tissage des filaments d'ordre multiplicatif (racines à la couche Lk–1) des bulbes identifiés en (a).
3. ...
4. Construction à partir du dernier caractère.

2.5.5 Exemples de construction de rhizome

Premier tour pour une couche de *O:M:(U:+B:). + M:O:(A:+T:).**

1. Identification des bulbes de couche L1 en relation d'addition livrés par le script.
   °O:M:(U:+B:). + M:O:(A:+T:).
   °O:M:(U:+B:).
   °M:O:(A:+T:).
2. Tissage des filaments d'ordre et de symétrie additive de couche L1 entre les bulbes identifiés en 1.
   °O:M:(U:+B:). + M:O:(A:+T:). $\oplus >$ [°O:M:(U:+B:). $\oplus\sqcap$ °M:O:(A:+T:).]
3. Tissage des filaments d'ordre multiplicatif (racines à la couche L0) des bulbes identifiés en 1.
   °O:M:(U:+B:). $\otimes >$ [°O: $\otimes$ °M: $\otimes$ °(U:+B:)]
   °M:O:(A:+T:). $\otimes >$ [°M: $\otimes$ °O: $\otimes$ °(A:+T:)]

Construction à partir du bulbe °O:M:(U:+B:).

1. Construction à partir du premier caractère
   (a) Identification des caractères en relation additive *contenus* dans le premier caractère.
   U: et A:
   identification des bulbes correspondants.
   °U:M:(U:+B:). et °A:M:(U:+B:).
   (b) Tissage des filaments d'ordre et de symétrie additive de couche L1 entre les bulbes identifiés en (a).
   °O:M:(U:+B:). $\oplus >$ [°U:M:(U:+B:). $\oplus\sqcap$ °A:M:(U:+B:).]

(c) Tissage des filaments d'ordre multiplicatif (racines à la couche L0) des bulbes identifiés en (a).
°U:M:(U:+B:). ⊗ > [°U: ⊗ °M: ⊗ °(U:+B:)]
°A:M:(U:+B:). ⊗ > [°A: ⊗ °M: ⊗ °(U:+B:)]

2. Construction à partir du second caractère

(a) Identification des caractères en relation additive *contenus* dans le second caractère.
S: B: et T:
Identification des bulbes correspondants.
°O:S:(U:+B:).
°O:B:(U:+B:).
°O:T:(U:+B:).

(b) Tissage des filaments d'ordre et de symétrie additive de couche L1 entre les bulbes identifiés en (a).
°O:M:(U:+B:). ⊕ > [°O:S:(U:+B:). ⊕⊓ °O:B:(U:+B:). ⊕⊓ °O:T:(U:+B:).]

(c) Tissage des filaments d'ordre multiplicatif (racines à la couche L0) des bulbes identifiés en (a).
°O:S:(U:+B:). ⊗ > [°O: ⊗ °S: ⊗ °(U:+B:)]
°O:B:(U:+B:). ⊗ > [°O: ⊗ °B: ⊗ °(U:+B:)]
°O:T:(U:+B:). ⊗ > [°O: ⊗ °T: ⊗ °(U:+B:)]

3. Construction à partir du troisième caractère

(a) Identification des caractères en relation additive *contenus* dans le troisième caractère.
U: et B:
Identification des bulbes correspondants.
°O:M:U:.
°O:M:B:.

(b) Tissage des filaments d'ordre et de symétrie additive de couche L1 entre les bulbes identifiés en (a).
°O:M:(U:+B:). ⊕ > [°O:M:U:. ⊕⊓ °O:M:B:.]

(c) Tissage des filaments d'ordre multiplicatif (racines à la couche L0) des bulbes identifiés en (a).
°O:M:U:. ⊗ > [°O: ⊗ °M: ⊗ °U:]
°O:M:B:. ⊗ > [°O: ⊗ °M: ⊗ °B:]

Construction à partir du premier caractère de °I:O:.

1. Ordres et symétries additives de couche 0 pour le premier caractère
°I: ⊕ > [°E: ⊕⊓ °F:]
°F: ⊕ > [°O: ⊕⊓ °M:]
°O: ⊕ > [°U: ⊕⊓ °A:]
°M: ⊕ > [°S: ⊕⊓ °B: ⊕⊓ °T:]
2. Bulbes correspondants de couche L1

(a) °E:O:. ⊗ > [°E: ⊗ °O: ⊗ °E:]
(b) °U:O:. ⊗ > [°U: ⊗ °O: ⊗ °E:]
(c) °A:O:. ⊗ > [°A: ⊗ °O: ⊗ °E:]
(d) °O:O:. ⊗ > [°O: ⊗ °O: ⊗ °E:]
(e) °S:O:. ⊗ > [°S: ⊗ °O: ⊗ °E:]
(f) °B:O:. ⊗ > [°B: ⊗ °O: ⊗ °E:]
(g) °T:O:. ⊗ > [°T: ⊗ °O: ⊗ °E:]
(h) °M:O:. ⊗ > [°M:⊗ °O: ⊗ °E:]
(i) °F:O:. ⊗ > [°F: ⊗ °O: ⊗ °E:]

3. Tissage des filaments d'ordre et de symétrie additive entre les bulbes de couche L1

(a) °O:O:. ⊕ > [°U:O:. ⊕⊓ °A:O:.]
(b) °M:O:. ⊕ > [°S:O:. ⊕⊓ °B:O:. ⊕⊓ °T:O:.]
(c) °F:O:. ⊕ > [°O:O:. ⊕⊓ °M:O:.]
(d) °I:O:. ⊕ > [°E:O:. ⊕⊓ °F:O:.]

4. Tissage des filaments d'ordre multiplicatif (racines à la couche L0)

(a) °E:O:. ⊗ > [°E: ⊗ °O: ⊗ °E:]
(b) °U:O:. ⊗ > [°U: ⊗ °O: ⊗ °E:]
(c) °A:O:. ⊗ > [°A: ⊗ °O: ⊗ °E:]
(d) °O:O:. ⊗ > [°O: ⊗ °O: ⊗ °E:]
(e) °S:O:. ⊗ > [°S: ⊗ °O: ⊗ °E:]
(f) °B:O:. ⊗ > [°B: ⊗ °O: ⊗ °E:]
(g) °T:O:. ⊗ > [°T: ⊗ °O: ⊗ °E:]
(h) °M:O:. ⊗ > [°M:⊗ °O: ⊗ °E:]
(i) °F:O:. ⊗ > [°F: ⊗ °O: ⊗ °E:]

Chapitre 3

Sémantique

« La mort et la vie sont au pouvoir de la langue. » Proverbes 18:20-21

Nous allons maintenant apprendre comment on produit (et reconnaît) des mots, des phrases, des textes et des hypertextes en IEML. La section 3.1 présente les circuits paradigmatiques du dictionnaire et la section 3.2 suivante présente les circuits syntagmatiques, textuels et hypertextuels et leurs formes de construction régulières.

3.1 Les circuits paradigmatiques du dictionnaire

3.1.1 Quelques définitions

3.1.1.1 Le dictionnaire

Le dictionnaire IEML est constitué d'un ensemble de réseaux paradigmatiques interopérables appelés *clés*. Les noeuds de ces réseaux représentent les termes du dictionnaire et les liens de ces réseaux représentent les relations sémantiques entre les termes.

3.1.1.2 Différence entre rhizome et circuit sémantique

Les deux principales différences entre un *circuit sémantique* et un *rhizome* sont que, premièrement, un circuit peut contenir ou canaliser des flux de courant sémantique et que, deuxièmement, un circuit se décrit automatiquement en relations sémantiques entre des expressions en langues naturelles.

3.1.1.3 Transcodage d'un rhizome en circuit sémantique

On peut comparer le rhizome codé par un script au plan en pointillé d'un ensemble de circuits possibles. Grâce à l'information fournie par le dictionnaire IEML, l'algorithme de transcodage du Rhizome vers la Sphère sémantique (voir 3.3) transforme certains bulbes d'un rhizome en *réservoirs*, certains filaments

de ce même rhizome en *canaux* et décrit les réservoirs et les canaux en langues naturelles.

3.1.1.4 Complexité des circuits sémantiques

Un script muni du dictionnaire peut donc générer automatiquement un circuit sémantique complexe qui articule notamment des circuits paradigmatiques, des circuits syntagmatiques, des circuits textuels et des circuits inter-textuels.

Le but de ce chapitre est d'expliquer la méthode de construction des circuits paradigmatiques du dictionnaire, qui permettent de donner sens automatiquement aux scripts IEML.

3.1.2 Méthode de construction des clés

Les circuits paradigmatiques sont construits à partir de circuits élémentaires appelés clés paradigmatiques, ou simplement *clés*. Nous allons d'abord expliquer comment on construit des *clés* en suivant une méthode semi-automatique, puis comment assembler ces clés en trousseaux cohérents.

3.1.2.1 Structure générale de la méthode de construction de clés

Pour construire une clé selon la méthode semi-automatique...

1. on applique un algorithme de formatage de clé à un script représentant une catégorie de couche Ln,
2. on sélectionne les filaments à *ne pas* transformer en canaux et les bulbes à *ne pas* transformer en réservoirs,
3. on décrit les réservoirs en langues naturelles, ce qui en fait des *termes*.

3.1.2.2 Algorithme de construction de clé

L'algorithme de construction de clé est semblable à l'algorithme de construction de rhizome (voir la section 2.5), avec ces deux différences :

1. il est limité à la couche Ln [1],
2. il transforme les bulbes en réservoirs et les filaments en canaux.

3.1.2.3 Sélection des filaments et des bulbes à ne pas ouvrir

Par défaut, tous les filaments sont transformés en canaux et tous les bulbes sont transformés en réservoirs. Pour obtenir la clé désirée, il suffit de sélectionner les bulbes et filaments à *ne pas* ouvrir.

1. C'est-à-dire au point 1 de la section 2.5.1.

CANAUX	**Multiplication** $\otimes$	**Addition** $\oplus$
Ordre $>$	$\otimes >$ relation étymologique	$\oplus >$ relation taxinomique
Symétrie $\sqcap$	$\otimes\sqcap$ relation de complémentarité	$\oplus\sqcap$ relation de concurrence

TABLE 3.1 – Les quatre canaux paradigmatiques entre les termes des clés originales

Restriction des canaux Concernant les filaments, il n'est possible d'annuler *que* la transformation des filaments *multiplicatifs* en canaux.

1. Pour les filaments de *symétrie* multiplicative on se contente d'annuler la transformation pour l'ensemble de la clé (voir la section 2.5.2 au point 6).
2. Pour les filaments *d'ordre* multiplicatif on choisit un, deux ou trois *rôles* (substance, attribut, mode) pour lesquels la transformation n'a pas lieu, et cela pour l'ensemble de la clé.

Restriction des réservoirs On sélectionne les bulbes à ne pas transformer en réservoirs simplement en les listant. Si le bulbe sélectionné pour ne pas être transformé en réservoir se trouve sur le chemin d'une relation d'ordre additif entre réservoirs, cette relation "saute" ou "traverse" le bulbe éliminé sous la forme d'un canal d'ordre additif. En revanche, la non-transformation du bulbe en réservoir interrompt les canaux multiplicatifs qui auraient pu passer par lui.

3.1.2.4 Relations entre termes des clés originales

Lorsqu'ils sont décrits en langues naturelles, les réservoirs des clés sont appelés des *termes* et les canaux représentent alors les *relations paradigmatiques* entre les termes. Les quatre types de relations paradigmatiques entre les termes des clés originales sont résumées dans le tableau 3.1.

1. Les relations d'ordre multiplicatif sont interprétés comme des relations *étymologiques*. Les termes de couche Ln–1 représentent alors les "racines" des termes de couche Ln. Le sens des termes de couche Ln dérive du sens de ses racines de couche inférieure, même si on ne peut jamais *déduire* le sens des termes de celui de leurs racines. Par exemple *s.u.-m.u.-'** (rêve) dérive du sens de *s.u.-** (image) et du sens de *m.u.-** (symptôme).
2. Les relations de symétrie multiplicative indiquent des *complémentarités* sémantiques. Par exemple, *s.u.-m.u.-'** (rêve) et *m.u.-s.u.-'** (fantasme) sont complémentaires.
3. Les relations d'ordre additif signalent des relations *taxinomiques*, c'est-à-dire des relations d'ensembles à sous-ensembles. Par exemple *s.u.-m.u.-

'** (rêve) appartient à *M:M:.u.-M:M:.u.-'** (fonctions sémiotiques). Il faut noter que *M:M:.u.-M:M:.u.-'** contient 81 séquences singulières.

4. Finalement, les relations de symétrie additive indiquent des rapports de *concurrence*. On peut entendre cette notion de concurrence en deux sens. Premièrement, au sens où le choix d'un terme pour figurer dans un énoncé équivaut à l'élimination des termes concurrents. Deuxièmement au sens où les termes concurrents "concourent" ensemble à la définition du terme qui les rassemble. Par exemple, dans la clé *M:M:.O:M:.-**, le terme *m.a.-** (parent) a pour concurrents les autres termes de la colonne *m.O:M:.-** (affect actualisé) tels que *m.y.-** (intelligence émotionnelle) et *m.u.-** (symptôme) ainsi les termes de la rangée *M:M:.a.-** (fonction sociale) tels que *b.a.-** (conteur) et *f.a.-** (guérisseur).

3.1.2.5 Le trousseau de clés interopérables

Un circuit paradigmatique est interopérable s'il ne comporte qu'une seule description en langues naturelles du même terme et si la même description n'est pas donnée à des termes distincts. Le *lexique de base* est un trousseau de clés originales distinctes qui forment ensemble un circuit paradigmatique interopérable. Les clés du lexique de base ne communiquent entre elles que par des canaux d'ordre multiplicatif (étymologie) et des super-clés. Le circuit paradigmatique du lexique de base n'est pas nécessairement connexe.

Ce *lexique de base* comprend l'ensemble des clés interopérables qui sont transversales à tous les univers de discours et qui peuvent être utilisées partout.

3.1.3 La clé *O:O:.**

Lorsque l'on applique l'algorithme de formatage de clé décrit en 3.1.2.2, à *O:O:.**, on obtient automatiquement l'ensemble des neuf réservoirs : *wo. / wa. / wu. / we. / U:O:. / A:O:. / O:U:. / O:A:. / O:O:.**. On obtient également le circuit des quarante-huit canaux décrit plus bas. Dans le cas de *O:O:.**, il n'y a pas besoin de sélectionner des bulbes ou des filaments à ne pas ouvrir et, comme on va le voir, tous les réservoirs sont des termes décrits en langues naturelles. La figure 3.1 représente de manière compacte les principales relations entre les termes.

3.1.3.1 Canaux paradigmatiques

Les relations paradigmatiques sont automatiquement définies comme suit par l'algorithme de formatage de clé de la section 3.1.2.2, qui dérive lui-même de l'algorithme de construction de Rhizome à partir du Script expliqué à la section 2.5.

— Canaux d'ordre multiplicatif (étymologie)
wo. $\otimes >$ [U: $\otimes$ U: $\otimes$ E:]
wa. $\otimes >$ [U: $\otimes$ A: $\otimes$ E:]

	O:O:. GÉNÉRER	
	O:U:. générer vers le virtuel	O:A:. générer vers l'actuel
U:O:. générer à partir du virtuel	UUE wo. réfléchir	UAE wa. opérer
A:O:. générer à partir de l'actuel	AUE wu. percevoir	AAE we. reconstituer

FIGURE 3.1 – La clé *O:O:.** (Générer)

wu. ⊗ > [A: ⊗ U: ⊗ E:]
we. ⊗ > [A: ⊗ A: ⊗ E:]
U:O:. ⊗ > [U: ⊗ O: ⊗ E:]
A:O:. ⊗ > [A: ⊗ O: ⊗ E:]
O:U:. ⊗ > [O: ⊗ U: ⊗ E:]
O:A:. ⊗ > [O: ⊗ A: ⊗ E:]
O:O:. ⊗ > [O: ⊗ O: ⊗ E:]

Rappelons que chacune des neuf lignes qui précèdent représente trois canaux d'ordre multiplicatif, ce qui fait en tout vingt-sept canaux d'ordre multiplicatif (ou relations étymologiques).

— Canaux de symétrie multiplicative (complémentarité)
wu. ⊗⊓ wa.
U:O:. ⊗⊓ O:U:.
A:O:. ⊗⊓ O:A:.

La clé O:O:. comporte donc trois canaux de symétrie multiplicative. Sur la figure 3.1, les deux cases *wo.** et *we.** marquent "l'axe de symétrie" diagonal de la relation de symétrie multiplicative wu. ⊗⊓ wa.

— Canaux d'ordre additif (taxonomie)
U:O:. ⊕ > wo.
U:O:. ⊕ > wa
A:O:. ⊕ > wu.

A:O:. $\oplus >$ we.
O:U:. $\oplus >$ wo.
O:U:. $\oplus >$ wu.
O:A:. $\oplus >$ wa.
O:A:. $\oplus >$ we.
O:O:. $\oplus >$ U:O:.
O:O:. $\oplus >$ A:O:.
O:O:. $\oplus >$ O:U:.
O:O:. $\oplus >$ O:A:.

La clé comporte donc douze canaux d'ordre additif.

— Canaux de symétrie additive (concurrence)
[wo. $\oplus\sqcap$ wa.]
[wu. $\oplus\sqcap$ we.]
[wo. $\oplus\sqcap$ wu.]
[wa. $\oplus\sqcap$ we.]
[U:O:. $\oplus\sqcap$ A:O:.]
[O:U:. $\oplus\sqcap$ O:A:.]

La clé comporte donc six canaux de symétrie additive.

*O:O:.** constitue la plus simple de toutes les clés originales et l'on voit qu'elle comporte déjà neuf termes et quarante-huit canaux paradigmatiques, dont vingt-sept (les relations étymologiques) concernent des relations entre les termes de la clé et les termes d'une clé de couche inférieure (ici, la couche 0). Pour tous les exemples qui suivent, nous ne prendrons pas la peine de lister explicitement tous les canaux, parce que nous supposerons que le lecteur a compris leur mode de construction automatique. Nous donnerons des tables d'allure matricielle du type de 3.1 et nous nous contenterons de signaler les bulbes non transformés en réservoirs et les filaments non transformés en canaux. Ceci fait, nous concentrerons l'exposé, de la manière la plus synthétique possible, sur *les relations sémantiques entre les termes* qui sont modélisées par les canaux paradigmatiques.

3.1.3.2 Relations sémantiques entre les termes

J'appelle *terme singulier* un terme qui correspond à une seule séquence des six symboles élémentaires. Le coeur de la clé O:O:. se trouve dans la relation entre ses quatre termes singuliers, une relation qui esquisse un cycle génératif (voir la figure 3.2).

1. *wa.** (opérer) correspond à un passage ou à une transformation du virtuel vers l'actuel. On suppose ici que l'opération dont il s'agit part d'un plan ou d'une visée virtuelle ou mentale et qu'elle aboutit, se réalise ou se projette dans un domaine actuel, physique.
2. *wu.** (percevoir) correspond à une transformation inverse à celle de *wa.** dans laquelle une réalité actuelle est transférée sur un plan virtuel, dans l'esprit, sous la forme d'une image de l'actuel.

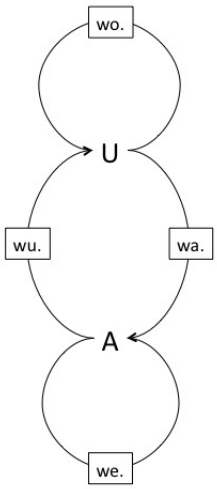

FIGURE 3.2 – Les 4 termes singuliers de *O:O:.**

3. *wo.** (réfléchir) correspond à une transformation ou un renouvellement interne à l'esprit. *wo.** régénère les virtualités, de telle sorte que l'ensemble du cycle soit effectivement productif ou créatif.
4. *we.** (reconstituer ou réagir) correspond à une transformation causale interne à l'environnement physique qui, en se reconstituant après l'action, répond d'une manière peut-être imprévisible, surprenante ou non-désirée à l'opération de *wa.**.

On peut comprendre *O:O:.** (générer) comme la structure fondamentale du cycle sensori *wu.** / moteur *wa.**, à condition d'admettre dans ce cycle les transformations internes au sujet pensant *wo** et à l'environnement physique *we.**. On pourra utiliser en IEML des verbes qui signifient non seulement l'une des quatre phases du cycle mais également...

1. les quatre phases ensemble : *O:O:.** (générer, en parcourant l'ensemble du cycle),
2. les deux phases qui prennent leur point de départ dans le virtuel : *U:O:.** (générer à partir du virtuel, c'est-à-dire "réfléchir et agir"),
3. les deux phases qui prennent leur point de départ dans l'actuel : *A:O:.** (générer à partir de l'actuel, c'est-à-dire "reconstituer et percevoir"),
4. les deux phases qui aboutissent dans le virtuel : *O:U:.** (générer vers le virtuel, c'est-à-dire "percevoir et réfléchir")
5. les deux phases qui aboutissent dans l'actuel : *O:A:.** (générer vers l'actuel, c'est-à-dire "opérer et reconstituer", un verbe qui inclut l'action et ses conséquences).

	O:M:. AGIR		
	O:S:. agir selon le signe	O:B:. agir selon l'être	O:T:. agir selon la chose
U:M:. agir virtuellement	USE y. savoir	UBE o. vouloir	UTE e. pouvoir
A:M:. agir actuellement	ASE u. énoncer	ABE a. s'engager	ATE i. faire

FIGURE 3.3 – La clé *O:M:.**

Le lecteur peut vérifier que les quarante-huit canaux paradigmatiques sont bel et bien fondés sur le plan des relations sémantiques entre termes.

3.1.4 La clé *O:M:.**

Comme dans le cas précédent, le circuit paradigmatique complet est obtenu automatiquement et sans restriction par l'algorithme décrit en 3.1.2.2 à partir de l'entrée *O:M:.**. Puisque sa substance est *O:**, comme la clé précédente, *O:M:.** est un verbe et ne contient que des verbes. Mais, cette fois-ci, l'attribut est *M:**, et c'est pourquoi l'objet ou le domaine de l'action sera plus précis.

3.1.4.1 Les six termes singuliers

Les six termes singuliers de la clé s'organisent ainsi :

Lorsque la substance est *U:**, comme dans *U:M:.**, il s'agit d'une virtualité ou d'une modalité de l'action [2]. On peut distinguer trois modalités :

y.* (savoir) correspond à l'attribut "signe" puisqu'il comprend des représentations ;

2. Voir, de Algirdas Julien Greimas et Joseph Courtès, *Sémiotique : dictionnaire raisonné de la théorie du langage*, Hachette, 1979 et de Algirdas Julien Greimas, *Du sens, 2*, Seuil, Paris, 1983

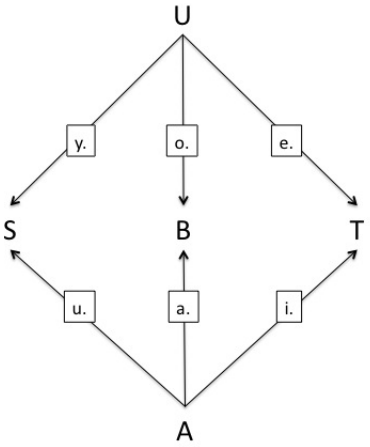

FIGURE 3.4 – Les six termes singuliers de *O:M:.**

o.* (vouloir) correspond à l'attribut "être" parce qu'il implique le noyau intentionnel de la personne ;

e.* (pouvoir) correspond à l'attribut "chose" étant donné qu'il se réfère à une capacité de transformer la réalité.

*U:M:.** (agir virtuellement) inclut les trois modalités et signifie donc : savoir, vouloir et pouvoir.

Lorsque la substance est *A:**, comme dans *A:M:.** il s'agit d'une action effective. De nouveau, on peut distinguer trois types d'actes effectifs :

u.* (énoncer) dont l'attribut est "signe" évoque l'expression, le dire, l'énonciation ;

a.* (s'engager), dont l'attribut est "être" signifie l'engagement, l'implication personnelle ;

i.* (faire), dont l'attribut est "chose" dénote l'action physique, l'acte matériel.

*A:M:.** (agir actuellement) comprend les trois types d'effectuation et signifie donc : énoncer, s'engager et faire.

*O:M:.** (agir en général) signifie alors : savoir, énoncer, vouloir, s'engager, pouvoir et faire. Toutes les dimensions de l'acte sont ici impliquées.

3.1.4.2 Les deux dialectiques

En général, les *symétries additives* des clés paradigmatiques (c'est-à-dire les *relations de concurrence* décrites en 3.1.2.4 au point 4), s'organisent en laissant invariant un sème en substance et en faisant varier le sème en attribut ou bien en

fixant un sème en attribut et en faisant varier le sème en substance. Ce procédé est illustré par la figure 3.4 dans laquelle le départ des flèches marque le sème en substance et leur arrivée le sème en attribut.

— **Dialectique signe / être / chose**
Si l'on fixe le sème en substance *U:** et que l'on fait varier le sème en attribut *S: / B: / T: **, alors on obtient avec "savoir / vouloir / pouvoir" une dialectique signe / être / chose. De même, si l'on fixe le sème en substance *A:** et que l'on fait varier le sème en attribut *S: / B: / T: **, alors on obtient de nouveau avec "énoncer / s'engager / faire" une dialectique signe / être / chose.

— **Dialectique virtuel / actuel**
En revanche, si l'on fixe le sème en attribut *S:** et que l'on fait varier le sème en substance *U: / A: **, alors on obtient avec *O:S:.** "savoir / énoncer" une dialectique virtuel / actuel. On obtient de la même manière une telle dialectique virtuel / actuel avec *O:B:.** "vouloir / s'engager" et *O:T:.** "pouvoir / faire".

3.1.5 La clé *E:F:.O:M:.**

Notre troisième et dernier exemple est fourni par la clé *E:F:.O:M:.-**. Contrairement aux deux précédentes clés, il ne s'agit pas d'un paradigme verbal mais d'un paradigme auxiliaire, puisqu'il commence par *E:**. Par contraste également avec les deux exemples qui précèdent, qui étaient à la couche L1, *E:F:.O:M:.-** se trouve à la couche L2. En outre, le circuit de ses canaux et de ses réservoirs est généré par l'algorithme décrit en 3.1.2.2 à partir de l'entrée *E:F:.O:M:.-**, mais avec cette restriction [3] qu'il n'existe pas de canaux d'ordre multiplicatifs (donc pas de relation étymologique avec des termes de couche immédiatement inférieure) correspondant aux sèmes *en substance*.

3.1.5.1 Invariance du sème en rôle de substance et variation du sème en rôle d'attribut

Nous allons d'abord considérer les huit cases sur la gauche de la figure 3.5, dans lesquels tous les termes ont invariablement *E:F:.** en substance. On se concentrera donc sur les variations significatives, qui concernent les sèmes en rôle d'attribut. Remarquons d'abord que l'attribut de *E:F:.O:M:.-** est *O:M:.**, qui vient juste d'être analysé en 3.1.4. Nous nous sommes servi de *y. / o. / e. / u. / a. / i.** pour indiquer le *mode* des verbes.

Puisque *y.** signifie "savoir", on l'utilisera pour signaler l'indicatif. Par exemple "savoir que l'on peut" s'écrira *e.-E:.-E:F:.y.-'** [4]. Utiliser l'indicatif en IEML consiste simplement à dire que l'on sait ce que l'on dit.

3. Voir 3.1.2.3 point 2.

4. Dans cet exemple, comme dans ceux qui suivent, la racine du verbe sera en rôle de substance, la conjugaison en rôle de mode et le rôle d'attribut sera vide. Voir en 3.2.4.2 la manière de construire des mots.

		E:F:.O:M:.- CONJUGAISON				
		E:O:.O:M:.- mode de participation		E:M:.O:M:.- temps / quand		
		E:U:.O:M:.- passif	E:A:.O:M:.- actif	E:S:.O:M:.- futur	E:B:.O:M:.- présent	E:T:.O:M:.- passé
E:F:.U:M:.- mode déclaratif	E:F:.y.- mode indicatif	E:U:.y.- connu	E:A:.y.- connaissant	E:S:.y.- connaîtra	E:B:.y.- connaît maintenant	E:T:.y.- a connu
	E:F:.o.- mode optatif	E:U:.o.- voulu	E:A:.o.- voulant	E:S:.o.- voudra	E:B:.o.- veut maintenant	E:T:.o.- voulait
	E:F:.e.- mode habilitatif	E:U:.e.- capable, habilité	E:A:.e.- capacitant, habilitant	E:S:.e.- pourra	E:B:.e.- peut maintenant	E:T:.e.- pouvait
E:F:A:M:.- mode performatif	E:F:.u.- mode expressif	E:U:.u.- exprimé	E:A:.u.- exprimant	E:S:.u.- exprimera	E:B:.u.- exprime maintenant	E:T:.u.- a exprimé
	E:F:.a.- mode promissif	E:U:.a.- engagé	E:A:.a.- engageant	E:S:.a.- s'engagera	E:B:.a.- s'engage maintenant	E:T:.a.- s'est engagé
	E:F:.a.- mode impératif	E:U:.i.- doit être fait	E:A:.i.- doit faire	E:S:.i.- devra exécuter	E:B:.i.- exécute maintenant	E:T:.i.- a dû exécuter devrait exécuter

FIGURE 3.5 – La clé *E:F:.O:M:.-**

Comme *o.** signifie “vouloir”, on l’utilisera pour signaler l’optatif, comme dans “vouloir réfléchir” ou “choisir de réfléchir” : *wo.-E:.-E:F:.o.-’**.

Etant donné que *e.** signifie “pouvoir”, on l’utilisera pour dénoter la capacité ou l’habilitation. Par exemple “pouvoir exprimer” se dira : *u.-E:.-E:F:.e.-’**.

Les modes indicatif, optatif et habilitatif se regroupent en un super-mode *E:F:.U:M:.-** baptisé déclaratif, ce qui indique simplement qu’il concerne les trois modalités (savoir, vouloir pouvoir) de l’action.

Le super-mode déclaratif est en relation de concurrence [5] avec un autre super-mode, dit performatif, qui regroupe 1) le mode expressif ou énonciatif basé sur *u.** (exprimer ou énoncer), 2) le mode promissif basé sur *a.** (s’engager) et 3) le mode impératif basé sur *i.** (faire). Dans le cas de ce dernier mode, c’est comme si le verbe à l’impératif était accompagné de la mention “à faire”. L’impératif implique évidemment la notion de devoir.

Même s’ils ne sont pas visibles sur la figure 3.5, d’autres regroupements en super-modes sont autorisés par l’algorithme de création de clé. Par exemple on peut utiliser le super-mode *E:F:.O:T:.-**, qui signifie “pouvoir *et* devoir”, comme dans *wa.-E:.-E:F:.O:T:.-’ ** (pouvoir et devoir opérer), et ainsi de suite.

3.1.5.2 Invariance du sème en rôle d’attribut et variation du sème en rôle de substance

Nous allons maintenant examiner les huit cases sur le haut de la figure 3.5. Ici, tous les termes ont invariablement *O:M:.** en rôle d’attribut et la variation significative concerne donc *E:F:.** en rôle de substance. Mais comme *E:** ne peut pas être décliné puisqu’il ne contient qu’une seule primitive, la variation ne peut porter en réalité que sur l’attribut de la substance, à savoir *F:**. Or nous savons que *F:** se décompose automatiquement en *O:** et *M:**.

Puisque *O:** représente une dialectique à deux pôles *U: / A:** et que nous sommes ici dans le contexte de la conjugaison, j’ai choisi de l’utiliser pour représenter la dialectique entre un mode passif *U:** et un mode actif *A:**. L’attribution de l’activité à l’actuel *A:** s’explique d’elle-même et il ne restait donc que le virtuel *U:** pour représenter le mode passif. On voit que la notion de “mode” s’entend ici en un sens perpendiculaire à celui qui était utilisé pour caractériser l’indicatif, l’optatif, l’habilitatif, l’expressif, le promissif et l’impératif. En effet, chacun de ces derniers modes peut être décliné en actif et passif. On peut par exemple être habilité, capable (passivité), *ou bien* habilitant, rendant capable (activité). J’ai donc appelé *E:O:.O:M:.-** “le mode de participation”, étant entendu que la participation peut être active ou passive.

Puisque *M:** représente une dialectique à trois pôles, il était presque inévitable, dans le contexte de la conjugaison, de s’en servir pour indiquer les trois principaux temps des verbes : le futur *S:**, le présent *B:** et le passé *T:**. J’ai d’abord attribué le passé à la “chose” puisque le passé est nécessairement réifié, et le présent à “l’être”. Il ne restait donc pour le futur que le “signe”.

5. Voir 3.1.2.4, point 4.

*E:M:.O:M:.-** représente donc “le temps”, c’est-à-dire la dialectique futur / présent / passé.

3.1.5.3 Matrice des termes singuliers

Les trente cases internes de la figure 3.5 contiennent toutes des termes singuliers et le sens de chaque terme découle automatiquement de la colonne (actif / passif / futur / présent / passé) et de la rangée (indicatif / optatif / habilitatif / expressif / promissif / impératif) à l’intersection desquelles il se trouve. On n’oubliera pas qu’il est toujours possible d’additionner plusieurs termes pour indiquer exactement la nuance que l’on a en tête. Par exemple : *wa.- E:.- (E:B:.a.- + E:T:.o.-)’** signifie : “avoir voulu et s’engager maintenant à opérer”.

3.1.5.4 Pour conclure cette liste d’exemples : procédés heuristiques

Les trois exemples que nous venons de passer en revue ont montré comment l’algorithme de construction automatique des clés se prête à l’arrangement sémantique des termes et de leurs relations paradigmatiques. Ils nous ont également permis de présenter les principaux procédés heuristiques de conception des clés et notamment :

1. l’utilisation créative des dialectiques E/F, O/M, U/A et S/B/T ;
2. les jeux sémantiques sur les variations dans un rôle syntaxique par invariance dans un autre rôle ;
3. la réutilisation des termes de couche Ln–1 comme sèmes des termes de couche Ln.

La figure 3.6 résume ce qu’il faut savoir des clés.

3.1.6 Le dictionnaire

3.1.6.1 Récapitulation des circuits paradigmatiques

En somme, il existe deux grands types de clés : les clés originales et les clés dérivées. Parmi les clés originales, on peut distinguer entre les clés ordinaires et les super-clés. Parmi les clés dérivées des clés originales, on distingue les clés analogiques et les clés spéciales. Le *dictionnaire* comprend aussi bien les clés originales que les clés dérivées (analogiques ou spéciales) et aussi bien le lexique de base que les thesaurii.

Les termes du dictionnaire sont formalisés par des réservoirs et les relations par des canaux. Ces termes entretiennent entre eux des relations d’ordre et de symétrie additive, multiplicative, analogiques et spécialisées.

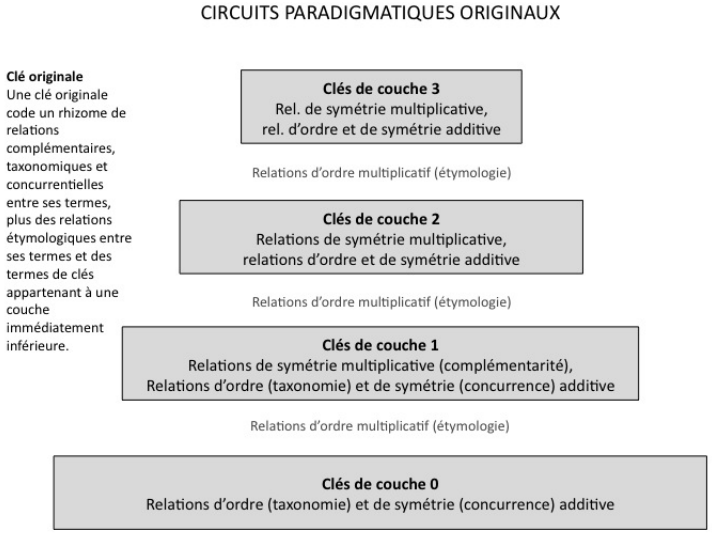

FIGURE 3.6 – Les clés originales

CANAUX	**Multiplication** $\otimes$	**Addition** $\oplus$
Ordre $>$	$\otimes >$ relation étymologique	$\oplus >$ relation taxinomique
Symétrie $\sqcap$	$\otimes\sqcap$ relation de complémenta-rité	$\oplus\sqcap$ relation de concurrence

TABLE 3.2 – Les 4 types de relations paradigmatiques

3.2 Les circuits énonciatifs

3.2.1 Les unités de sens

3.2.1.1 La notion d'unité de sens

Aussi bien dans les circuits paradigmatiques que dans les circuits syntagmatiques, textuels ou inter-textuels, on n'a plus affaire à des catégories et aux rhizomes correspondants (comme dans le Script avant toute interprétation en langues naturelles) mais à des *unités de sens*, c'est-à-dire à des réservoirs interconnectés par des canaux et décrits en langues naturelles selon des clés (que ces clés soient originales, transposées ou spécialisées).

3.2.1.2 Unités en langue et unités en parole

On peut distinguer deux grands types d'unités de sens :

1. les unités "en langue", qui sont les termes du dictionnaire interconnectés par des circuits *paradigmatiques* ;
2. les unités "en parole" qui sont les propositions, textes et hypertextes connectés respectivement par des circuits syntagmatiques, inter-propositionnels et inter-textuels. Les unités de sens "en parole" correspondent à des *circuits énonciatifs*, c'est-à-dire aux circuits syntagmatiques (propositions), inter-propositionnels (textes ou USL) et inter-textuels (hypertextes).

3.2.1.3 Termes

Comme nous l'avons vu au chapitre 3.1, les *termes* sont les unités de sens élémentaires qui servent de fondement à la construction de toutes les autres unités de sens. Le dictionnaire connecte les termes entre eux au moyen de canaux paradigmatiques qui décrivent leurs relations sémantiques "en langue" (voir 3.1.2.4 et 3.1.6.1 pour la sémantique de ces canaux).

3.2.1.4 Propositions

Description générale Les *propositions* décrivent les relations sémantiques entre termes *dans des énoncés*, c'est-à-dire "en parole". Dans les propositions, les termes sont additionnés pour produire des *morphèmes*, les morphèmes sont multipliés pour produire des *mots*, les mots sont multipliés pour produire des *clauses*, les clauses sont additionnées pour produire des *phrases*, et l'on peut ainsi produire des *phrases de degrés supérieurs*, jusqu'à ce que la limite de la couche L6 soit atteinte.

Les relations opératoires (additives et multiplicatives) récursives entre termes qui viennent d'être décrites commandent des circuits syntagmatiques. Les réservoirs de ces circuits syntagmatiques sont, respectivement, les morphèmes, les mots, les clauses et les phrases de degrés successifs, tandis que les canaux entre les réservoirs décrivent leurs relations grammaticales.

Unités propositionnelles	Construction par opérations isocouches
Morphème	(terme $i \oplus$ terme j...)
Mot	(morphème $i \otimes$ morphème j)
Clause de degré D1	(mot $i \otimes$ mot $j \otimes$ mot k)
Phrase de degré D1	(clause i D1 $\oplus$ clause j D1...)
Clause de degré Dn	(ph. i Dn–1 $\otimes$ ph. j Dn–1 $\otimes$ ph. k Dn–1)
Phrase de degré Dn	(clause i Dn $\oplus$ clause j Dn...)

Table 3.3 – Hiérarchie des unités propositionnelles

Détail de la hiérarchie des unités propositionnelles Le tableau 3.3 ci-dessous présente de manière synthétique la hiérarchie des unités propositionnelles.

- Un morphème se construit par une addition de termes.
- Un mot fléchi se construit par une multiplication entre un morphème racine et un morphème de flexion.
- Les clauses de degré D1, qui sont des multiplications de mots, décrivent la relation grammaticale entre un mot en rôle de substance et un mot en rôle d'attribut par le moyen d'un mot en rôle de mode.
- Les phrases de degré D1, qui sont des additions de clauses, décrivent un réseau grammatical de mots.
- Les clauses de degré D2, qui sont des multiplications de phrases, décrivent la relation grammaticale entre une phrase en rôle de substance et une phrase en rôle d'attribut par le moyen d'une phrase en rôle de mode, et ainsi de suite jusqu'aux phrases de dernier degré Dn.

Un morphème peut n'être constitué que d'un seul terme, un mot d'un seul morphème, une clause d'un seul mot, une phrase d'une seule clause et ainsi de suite. Une proposition peut donc évidemment n'être constituée que d'un seul mot.

Règle des séries vides En construisant les propositions, *on peut toujours additionner ou multiplier des unités de sens de couches différentes*, à condition d'ajouter des *séries vides* aux unités de sens des couches les plus basses, de manière à ce qu'elle soient ainsi promues aux couches auxquelles appartiennent les unités de sens les plus hautes. *Le sens d'une unité n'est pas modifié par l'ajout d'une série vide.*

Par exemple, si l'on veut dire "officier *l.a.-k.a.-f.o.-'** *et* juge *n.a.-**" on écrit :

*(n.a.-' + l.a.-k.a.-f.o.-')** en ajoutant une série vide à *n.a.-** afin de le promouvoir à la couche L3 [6].

6. Pour tous les exemples de cet ouvrage, il est suggéré de consulter le dictionnaire en ligne afin de pouvoir situer les termes dans les circuits de relations sémantiques où ils prennent sens.

3.2.1.5 Différences entre unités propositionnelles et unités supérieures

La principale différence entre unités de sens propositionnelles, d'une part, et unités de sens textuelles et hypertextuelles, d'autre part, est que les unités propositionnelles (morphèmes, mots, phrases) ne peuvent s'additionner ou se multiplier qu'avec des unités de même couche. En revanche, aucune contrainte *de couche* ne limite les additions entre propositions d'un même texte, ni les additions et les multiplications récursives de textes qui construisent les hypertextes.

3.2.1.6 Textes (USL)

Les *textes* sont des unités de sens qui résultent de relations additives entre propositions. Le texte peut être considéré comme le résultat de l'addition entre les propositions qu'il contient (chaque proposition est en relation d'ordre additif avec le texte qui la contient) et les différentes propositions d'un même texte sont, par défaut, en relations mutuelles de symétrie additive.

3.2.1.7 Hypertextes

Finalement, les *hypertextes* sont des unités de sens qui résultent d'opérations récursives d'addition et de multiplication entre textes. On se souvient que la multiplication sémantique est une opération à trois variables (substance, attribut, mode). Au niveau hypertextuel, le rôle de substance est joué par un ensemble de textes en référence (ou cités), le rôle d'attribut est joué par un ensemble de textes qui réfèrent aux textes en rôle de substance (ou citants) et le rôle de mode est joué par un ensemble de textes explicitant la relation entre les textes en référence et les textes qui s'y réfèrent (ou note de citation). La multiplication modélise donc ici l'acte de référence inter-textuel (ou de citation).

Les textes (USL) peuvent être assimilés à des unités de sens de degré hypertextuel 0 et l'on peut alors construire des "propositions hypertextuelles" de degré 1, 2, 3, 4, 5 et 6. Ces propositions peuvent être additionnées pour produire des textes hypertextuels d'ordre supérieur, ces textes hypertextuels peuvent à leur tour servir à de nouvelles constructions de circuits, et ainsi de suite.

En somme, comme il est indiqué dans le tableau 3.4, les hypertextes (circuits récursifs de textes) résultent de l'addition et de la multiplication d'USL de la même manière que les phrases sont construites récursivement à partir de mots.

Une multiplication d'USL aboutissant à une clause hypertextuelle de degré 1 sera de la forme :

*USL**:*USL**:*USL**:.

Une multiplication d'USL aboutissant à une clause hypertextuelle de degré 2 sera de la forme :

*USL**:*USL**:*USL**:. *USL**:*USL**:*USL**:. *USL**:*USL**:*USL**:.-

On peut ainsi monter les degrés multiplicatifs hypertextuels jusqu'au sixième, en utilisant les signes de ponctuation d'IEML (: . - ' , _ ;). Un USL d'USL - ou USL d'ordre 1 - est obtenu en additionnant des hypertextes de degrés 1 à 6. Rien n'empêche en principe de construire des USL d'ordre 2, 3, etc. jusqu'à n

Unités textuelles et hypertextuelles	Construction par opérations libres à partir d'USL d'ordre 0
Texte (USL)	(proposition $i \oplus$ proposition j...)
Clause hypertextuelle de degré D1	(USL $i \oplus$ USL j ...) $\otimes$ (USL $k \oplus$ USL l ...) $\otimes$ (USL $m \oplus$ USL n ...)
Hypertexte de degré D1	(clause hyp. i D1 $\oplus$ clause hyp. j D1...)
Clause hypertextuelle de degré Dn	(hyp. i Dn–1 $\oplus$ hyp. j Dn–1...) $\otimes$ (hyp. k Dn–1 $\oplus$ hyp. l Dn–1...) $\otimes$ (hyp. m Dn–1 $\oplus$ hyp. n Dn–1...)
Hypertexte de degré Dn	(clause hyp. i Dn $\oplus$ clause hyp. j Dn...)

Table 3.4 – Hiérarchie des unités textuelles et hypertextuelles

(n étant un entier naturel fini). Le tableau 3.4 se limite à décrire la hiérarchie hypertextuelle pour les USL d'ordre 0.

3.2.2 Les trois classes

Toutes les langues naturelles comportent différentes *classes* d'unités de sens telles que : noms, verbes, adverbes, adjectifs, prépositions, conjonctions, marqueurs de genre, de nombre, de cas, de temps, de mode, de personne, etc. Les classes d'unités verbales et nominales sont universelles. Il faut noter en outre qu'il existe non seulement des mots mais aussi des propositions qui sont verbales ou nominales. En IEML, il n'existe que trois classes d'unités dans les circuits paradigmatiques et syntagmatiques : les unités verbales, nominales et auxiliaires, mais ces classes peuvent traduire toutes celles des langues naturelles.

3.2.2.1 Classe verbale

Les unités de sens qui commencent par *O:** (*U:** ou *A:**) sont *verbales*. Il peut s'agir de mots-verbes ou de phrases verbales. Les unités verbales indiquent des processus, des habitus ou des événements.

3.2.2.2 Classe nominale

Les unités *nominales* commencent par *M:** (*S:**, *B:** ou *T:**). Elles indiquent des entités ou des qualités. Elles comprennent des noms ou des adjectifs, ainsi que des propositions nominales.

3.2.2.3 Classe auxiliaire

Finalement, les unités qui commencent par *E:** sont des unités *auxiliaires*. Elles comprennent les prépositions, pronoms, adverbes, conjonctions, marqueurs de conjugaison, de mode, de cas, de genre, de nombre, de personne, etc. Placées en rôle de mode, les auxiliaires modifient les morphèmes pour former des mots fléchis ou indiquent la relation entre l'unité substance et l'unité attribut dans

une phrase. Les *phrases auxiliaires* en rôle de mode représentent la relation entre les phrases en rôle de substance et les phrases en rôle d'attribut.

3.2.2.4 Interprétation des classes selon leur rôle syntaxique

Des termes verbaux ou nominaux peuvent être utilisés en rôle de mode pour indiquer le "cas sémantique" d'un mot.

On trouvera ci-dessous un exemple de terme verbal utilisé en rôle de mode pour "décliner" le sens d'un terme nominal en rôle de substance. *n.o.-n.o.-'** signifie : "être humain". [7]

n.o.-n.o.-'E:.-'wu.wo.-',* "humain enfant" (à partir du verbe *wu.wo.-** "être un enfant")

n.o.-n.o.-'E:.-'wu.wa.-',* "humain jeune" (à partir du verbe *wu.wa.-** "être jeune")

n.o.-n.o.-'E:.-'wu.wu.-',* "humain adulte" (à partir du verbe *wu.wu.-** "être mûr")

n.o.-n.o.-'E:.-'wu.we.-',* "humain vieux" (à partir du verbe *wu.we.-** "être vieux")

De même, des unités verbales ou nominales utilisées en rôle de mode dans une phrase peuvent indiquer le type de relation entre l'unité substance et l'unité attribut.

Lorsqu'elles sont placées en substance ou en attribut, les unités de sens auxiliaires peuvent être prises en un sens nominal ou verbal selon le contexte grammatical indiqué par le mode. Nous avons déjà rencontré cette situation plus haut. Dans :

*E:T:.c.-' m.a.-' E:E:U:.-', E:S:.wo.-', E:E:A:.-',_** "Etat du Moi Parent"

l'auxiliaire *E:T:.c.-** "état" utilisé en rôle de substance, est pris en un sens nominal puisque la construction grammaticale implique que *m.a.-'** "parent" qualifie *E:T:.c.-**. En effet, *E:E:U:.** marque l'attribut du nom.

3.2.2.5 Règle de décision pour attribuer une unité syntagmatique ou paradigmatique à l'une des trois classes

Le premier caractère, ou *l'initiale*, d'une chaine de caractères peut être *mixte*, si ses primitives n'appartiennent pas exclusivement à l'une des trois classes. Pour résoudre ce problème, la *règle de décision* permettant d'attribuer une unité de sens syntagmatique à l'une des trois classes est la suivante :

`E`+`O` $\Longrightarrow$ `O`
`E`+`M` $\Longrightarrow$ `M`
`O`+`M` $\Longrightarrow$ `M`

Par exemple, *(E:+U:+B:)** sera considérée comme une unité nominale. Les mêmes règles sont appliquées pour déterminer la classe d'un paradigme lorsqu'il résulte de l'addition d'un script à initiale O et d'un script à initiale M

7. Dans ce cas, le rôle attribut est occupé par une séquence vide voir plus bas, à ce sujet, la section 3.2.4.2.

3.2.3 Les trois rôles

3.2.3.1 Introduction aux trois rôles

Contrairement aux trois classes d'unités de sens (verbe, nom et auxiliaire), les trois rôles syntaxiques d'IEML (substance, attribut et mode) n'indiquent pas la *nature* des unités de sens mais leur *fonction* grammaticale, et nous venons de voir que la fonction a préséance sur la nature. En règle générale, une unité, quelle que soit sa classe, peut jouer n'importe lequel des trois rôles dans une unité de couche supérieure.

Quel est le sens de ces « places » qui se répètent et s'emboîtent fractalement les unes dans les autres et qui renvoient aux trois variables de la multiplication sémantique ? Nos trois rôles décrivent fondamentalement *les trois facteurs d'un acte de prédication* : "ceci" (l'attribut) est prédiqué de "cela" (la substance) de "cette manière" (le mode).

Considérons par exemple la phrase "un étudiant apprend les mathématiques"
*
(y.a.-' b.-j.-s.y.-' E:E:T:.-'
+
y.a.-' d.a.-s.a.-f.o.-' E:E:S:.-')
**

Termes

E:E:S:. nominatif (sujet)

E:E:T:. accusatif (complément d'objet)

y.a.- apprendre

b.-j.-s.y.-' mathématiques

d.a.-s.a.-f.o.-' étudiant

1. La première clause (*y.a.-' b.-j.-s.y.-' E:E:T:.-'*) dit que "les mathématiques" sont prédiquées du verbe "apprendre" sur le mode de "l'objet".
2. La seconde clause (*y.a.-' d.a.-s.a.-f.o.-' E:E:S:.-'**) dit que "l'étudiant" est prédiqué du verbe "apprendre" sur le mode du "sujet".

L'exemple qui précède illustre les trois rôles *dans les circuits syntagmatiques*. Venons-en maintenant aux trois rôles *dans les circuits inter-textuels*. Nous avons vu plus haut dans notre discussion des hypertextes (voir 3.2.1.2, point 4) qu'au dessus du niveau du texte, les trois rôles s'interprètent selon le schéma : "texte auquel on se réfère" (substance) / "texte qui se réfère" (attribut) / "note de référence" (mode).

Aussi bien dans les circuits syntagmatiques que dans les circuits inter-textuels, les trois rôles construisent une *boucle prédicative auto-référentielle* dans laquelle la signification de chacune des unités qui joue l'un des rôles dépend de la signification des unités qui jouent les deux autres rôles.

En fin de compte, les trois rôles renvoient aux triplets (sommet d'arrivée, sommet de départ, étiquette de lien) de la théorie des graphes. C'est justement pourquoi les opérations sémantiques d'IEML peuvent se traduire automatiquement sous forme de graphes. Une multiplication peut se lire comme la

construction d'une arête entre deux sommets et une addition de multiplications comme la construction d'un *ensemble* d'arêtes entre des sommets, c'est-à-dire comme la construction d'un graphe. A la couche ou au degré supérieurs [8], un graphe devient lui-même un sommet qui sera connecté à d'autres sommets par de nouvelles arêtes.

3.2.3.2 Le rôle substance

La substance est ce dont on prédique quelque chose, ou "le sujet" de la prédication au sens aristotélicien du terme. Elle détermine (1) ce dont on parle et (2) l'orientation grammaticale d'une expression. Dans les circuits inter-textuels, le rôle substance désigne le texte cité ou en référence.

Si le premier caractère, ou l'*initiale*, d'une substance est de type *O:** alors la substance est un verbe ou la racine verbale d'un mot et toute la séquence est elle-même un verbe ou une phrase verbale. Si l'initiale d'une unité substantielle est de type *M:**, alors la substance est un nom, la racine nominale d'un nom ou d'un adjectif et la totalité de la catégorie est elle-même un nom ou une phrase nominale. Nous avons vu enfin que les adverbes, prépositions, conjonctions, pronoms et inflexions diverses, telles que les cas et conjugaisons, commencent par *E:**. L'initiale de leur substance est "vide".

Il est donc clair que la notion de substance, au sens qu'elle a dans le vocabulaire technique d'IEML, ne doit pas être interprétée comme excluant les notions de qualité, de processus, d'événement, de manière ou de relation. En effet, le rôle substance de la prédication peut être joué par une unité de sens appartenant à n'importe laquelle des trois classes.

3.2.3.3 Le rôle attribut

L'attribut est une qualité, un trait distinctif, un facteur actualisant (comme le sujet d'un verbe) ou un complément de la substance. Dans les circuits inter-textuels, le rôle attribut désigne *le texte citant* qui construit le texte cité comme une référence.

Dans les termes du dictionnaire, la substance indique la principale notion de la sémantique d'un terme tandis que l'attribut indique une variation sur la notion principale ou une propriété supplémentaire.

Dans les phrases verbales, l'attribut peut être le sujet, l'objet ou n'importe quelle sorte de complément du verbe qui joue le rôle substantiel. La fonction grammaticale précise de l'attribut (autrement dit, la relation entre la substance et l'attribut) est indiquée par le mode.

Dans les phrases nominales, l'attribut peut être « l'attribut » du nom (au sens grammatical ordinaire du terme), un génitif ou n'importe quel complément du nom qui joue le rôle substantiel. De nouveau, la fonction grammaticale précise de l'attribut est notée par le mode.

8. La couche pour les circuits syntagmatiques et le degré pour les circuits inter-textuels

3.2.3.4 Le rôle mode

Dans les termes du dictionnaire, le mode est utilisé pour indiquer une variante sur la signification d'une racine terminologique, la racine étant constituée par le groupe substance-attribut.

Dans la construction des mots, le mode est utilisé pour marquer les flexions telles que le genre, le nombre, la personne (première, deuxième, troisième), les modes et temps des verbes, les cas des noms, etc.

Dans la construction des phrases (sachant qu'il existe des phrases de phrases, etc.), le mode est utilisé pour définir la *relation* grammaticale entre l'unité en rôle substance et l'unité en rôle attribut.

En 2013, le dictionnaire IEML contenait déjà plus de 150 cas grammaticaux, conjugaisons, pronoms, adverbes et prépositions [9] de couche 1, 2 et 3. Ces termes auxiliaires (qui ne sont ni des noms ni des verbes) ont été principalement conçus *pour être utilisés en rôle modal.* Rappelons cependant que rien n'empêche d'utiliser des unités nominales ou verbales en rôle de mode ou les termes auxiliaires en rôle de substance ou d'attribut.

Dans les circuits inter-textuels, le rôle mode désigne *la note de citation ou de référence* qui connecte le texte citant au texte cité (en référence).

3.2.4 Des morphèmes aux propositions

3.2.4.1 Morphèmes IEML

Un morphème est un terme, ou une addition de termes promus à la même couche, qui figurent dans un circuit syntagmatique.

Par exemple, dans :

y.-E:.-(E:S:.O:M:.- + E:S:.wo.-)'

qui signifie "je saurai", *y.** et *(E:S:.O:M:.- + E:S:.wo.-)** constituent des morphèmes.

Termes

E:S:.O:M:.- futur

E:S:.wo.- première personne du singulier (je)

y. savoir

3.2.4.2 Mots IEML

Il existe deux grands types de mots en IEML : les mots-morphèmes et les mots fléchis.

Mots-morphèmes Les mots-morphèmes ou mots non-fléchis *ne comportent pas* de cas, genre, nombre, conjugaison, articles, prépositions, etc. Par exemple, *y.** "savoir" ou *(y. + e.)* "savoir et pouvoir" sont des mots-morphèmes.

9. Post-position serait plus adéquat, puisque le mode vient *après* ce qu'il modifie

Mots fléchis Dans les mots fléchis, le morphème racine est placé en rôle de substance, le morphème de flexion est placé en rôle de mode et le rôle attribut est occupé par une série vide.

Premier exemple:

y.-E:.-(E:U:A:.- + E:S:.O:M:.- + E:T:.wo.-)'

signifie : "Ils ne sauront pas." et constitue un mot.

Termes

E:U:A:. négation

y. savoir

E:S:.O:M:.- futur

E:T:.wo.- troisième personne du pluriel (ils)

Deuxième exemple:

*m.i.-k.i.-'E:.-'E:U:.wa.-',**

signifie : "ces jeux" et constitue un mot.

Termes

m.i.-k.i.-' jeu

E:U:.wa.- pronom démonstratif pluriel

3.2.4.3 Clauses IEML

Les clauses de degré 1 sont des multiplications d'additions de mots dans lesquelles l'unité en rôle de mode explicite la relation grammaticale que l'unité en rôle d'attribut entretient avec l'unité en rôle de substance.

Par exemple,

*

(u.-E:.-(E:B:.y.- + E:T:.wo.-)' b.a.-'E:E:S:.-',

+

u.-E:.-(E:B:.y.- + E:T:.wo.-)' t.o.-b.o.-'E:E:T:.-',)

**

est une phrase qui s'analyse en deux clauses :

1. La première clause contient trois mots : "un conteur" *b.a.-** "considéré comme sujet" *E:E:S:.-** "raconte" *u.-E:-(E:B:.y.-+ E:T:.wo.-)'**
2. La seconde clause contient trois mots : "raconte" *u.-E:-(E:B:.y.-+ E:T:.wo.-)'**, "une histoire" *t.o.-b.o.-'** "considérée comme objet" *E:E:T:.-**

L'ensemble de la phrase signifie donc : "Un conteur raconte une histoire".

Termes

E:E:S:. nominatif (sujet)

E:E:T:. accusatif (complément d'objet)

u. énoncer (traduit "raconter" dans la phrase parce que l'objet est une histoire)

E:B:.y.- présent de l'indicatif

E:T:.wo.- troisième personne du singulier

b.a.- conteur

t.o.-b.o.-' histoire, récit

3.2.4.4 Phrases IEML

Les phrases de niveau 1 sont des additions de clauses de degré 1, les clauses de degré 2 sont des multiplications de phrases de degré 1, les phrases de degré 2 sont des additions de clauses de degré 2 et ainsi de suite.

Exemple 1 "L'enfant de ma voisine est souvent autorisé par un médecin à ne pas aller à l'école"

Cette phrase compte quatre clauses de couche L5, qui se succèdent évidemment selon l'ordre du script.

*
(u.A:.- E:.- (E:U:A:.- + E:U:.e.- + E:B:T:.-)', h.y.-', E:T:.f.-',_
+
u.A:.- E:.- (E:U:A:.- + E:U:.e.- + E:B:T:.-)', n.a.-f.a.-f.o.-', E:B:.x.-',_
+
u.A:.- E:.- (E:U:A:.- + E:U:.e.- + E:B:T:.-)', d.a.-m.a.-f.o.-' E:.-' E:A:.p.-', E:E:S:.-',_
+
d.a.-m.a.-f.o.-' E:.-' E:A:.p.-', m.a.-k.a.-f.o.-' E:.-' (E:A:.wo.-' + E:S:.wo.-' + E:F:.we.-'), E:E:A:.-', _)
**

Termes

E:E:A:. génitif

E:E:S:. nominatif

E:U:A:. négation

E:B:T:. souvent

E:U:.e.- mode passif abilitatif

E:A:.wo.- possessif singulier

E:A:.p.- article défini

E:S:.wo.- je, moi (première personne du singulier)

E:B:.x.- agent

E:T:.f.- dans (entrer dans)

u.A:.- aller

h.y.- école

E:F:.we.- genre féminin

m.a.-k.a.-f.o.-' voisin

n.a.-f.a.-f.o.-' médecin

d.a.-m.a.-f.o.-' enfant

Traduction de la phrase clause par clause

1. Commençons par :
   u.A:.- E:.- (E:U:A:.- + E:U:.e.- + E:B:T:.-)', h.y.-', E:T:.f.-',_
   u.A:.- E:.- (E:U:A:.- + E:U:.e.- + E:B:T:.-)' signifie "souvent autorisé (habilité) à ne pas aller" où "aller" *u.A:.-** est la racine du verbe et *(E:U:A:.- + E:U:.e.- + E:B:T:.-)** sont ses trois flexions. *****h.y.-** est l'école et *****E:T:.f.-** montre que l'école est le complément de lieu du verbe.
2. **u.A:.- E:.- (E:U:A:.- + E:U:.e.- + E:B:T:.-)', n.a.-f.a.-f.o.-', E:B:.x.-',_**
   Ici le verbe au mode passif est le même que dans la clause précédente et *n.a.-f.a.-f.o.-',** (le médecin) est indiqué comme complément d'agent *E:B:.x.-',**.
3. **u.A:.- E:.- (E:U:A:.- + E:U:.e.- + E:B:T:.-)', d.a.-m.a.-f.o.-' E:.-' E:A:.p.-', E:E:S:.-',_**
   Dans cette troisième clause, le verbe est encore le même et c'est le sujet *E:E:S:.** qui est indiqué, à savoir l'enfant (racine *d.a.-m.a.-f.o.-'** et flexion *E:A:.p.-** pour marquer l'article défini).
4. **d.a.-m.a.-f.o.-' E:.-' E:A:.p.-', m.a.-k.a.-f.o.-' E:.-' (E:A:.wo.-' + E:S:.wo.-' + E:F:.we.-'), E:E:A:.-', _**
   Dans cette quatrième et dernière clause, la substance est jouée par "l'enfant" *d.a.-m.a.-f.o.-' E:.-' E:A:.p.-',** et l'enfant en question est qualifié : c'est celui de (génitif *E:E:A:.-**) "ma voisine" *m.a.-k.a.-f.o.-' E:.-' (E:A:.wo.-' + E:S:.wo.-' + E:F:.we.-'),**. La flexion *(E:A:.wo.-' + E:S:.wo.-' + E:F:.we.-')** marque simultanément le possessif singulier, la première personne du singulier et le féminin, donc *ma* voisine.

Les sèmes en rôles de substance et d'attribut des clauses de la même phrase sont considérées comme les sommets d'un graphe. Un sème en rôle de mode est considéré comme l'étiquette de l'arête connectant le sommet-sème en rôle de substance et le sommet-sème en rôle d'attribut. Cette étiquette explicite la relation grammaticale entre les deux sommets. C'est ainsi que la phrase qui vient d'être analysée, comme toutes les phrases IEML, peut être représentée comme un *réseau sémantique,* ainsi qu'on peut le voir sur la figure 3.7, où les verbes et les noms sont représentés sur un fond gris et les auxiliaires sur un fond blanc.

Exemple 2 Dans ce second exemple, nous allons illustrer le cas de deux phrases de niveau 1 (la proposition principale et la proposition subordonnée) connectées par une phrase en rôle de mode : "J'ai appris les mathématiques parce que j'ai suivi un bon professeur". L'ensemble de la proposition est donc une clause de niveau 2.

*
y.a.- E:.- (E:S:.wo.- + E:T:.e.-)' b.-j.-s.y.-' E:E:T:.-',_
a.o.- E:.- (E:S:.wo.- + E:T:.o.-)', d.a.-f.a.-f.o.-' E:.-' E:U:T:.-', E:E:T:.-',_
E:M:.B:O:.-',_;
**

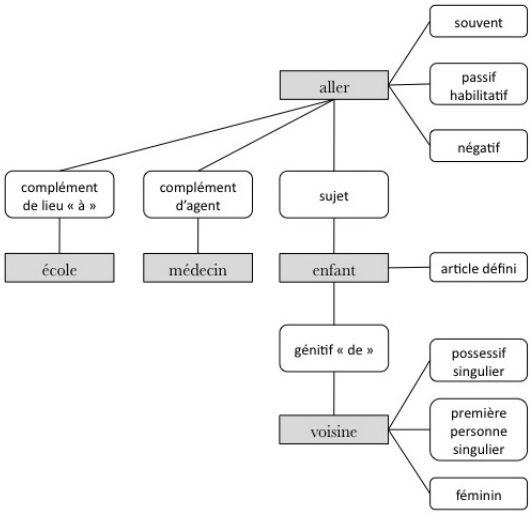

FIGURE 3.7 – L'enfant de ma voisine est souvent autorisé par un médecin à ne pas aller à l'école

Termes

E:E:T:. accusatif (complément d'objet direct)

E:U:T:. bon, meilleur

E:S:.wo.- je (première personne du singulier)

E:T:.o.- passé optatif

E:T:.e.- passé abilitatif

a.o.- suivre

y.a.- apprendre

E:M:.B:O:.- cause, rôle fonctionnel "parce que"

b.-j.-s.y.-' mathématiques

d.a.-f.a.-f.o.-' enseignant

1. La phrase substance ou, si l'on veut, la proposition principale signifie "j'ai appris les mathématiques". Le mode du verbe indique une nuance abilitative (j'ai *pu* apprendre).
2. La phrase attribut ou, si l'on veut, la proposition subordonnée signifie "J'ai suivi un bon enseignant". Le mode du verbe indique une nuance optative (j'ai *voulu* suivre). On notera que "bon" *E:U:T:.** fait office de flexion de la racine "enseignant" *d.a.-f.a.-f.o.-'**.
3. La phrase de mode (il s'agit ici évidemment d'une phrase-mot) indique la relation "parce que" entre la phrase substance et la phrase attribut.

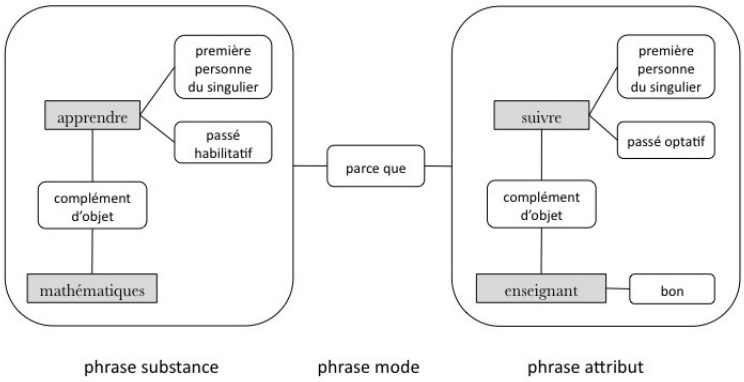

FIGURE 3.8 – J'ai appris les mathématiques parce que j'ai suivi un bon professeur

Le réseau sémantique correspondant à cette clause de niveau 2 est illustré sur la figure 3.8, où les noms et les verbes apparaissent sur fond gris et les auxiliaires sur fond blanc.

3.2.5 Des propositions aux textes (USL)

3.2.5.1 Stratégie d'écriture des USL

Il va sans dire que, pour l'utilisateur final qui ne voudra pas s'embarrasser de fabriquer des USL, il sera toujours possible d'utiliser des interfaces graphiques qui lui permettront de formuler des tags simples en langues naturelles, ces tags étant automatiquement traduits en IEML.

Rappelons pour commencer que tout USL représente évidemment *une interprétation possible* de l'objet, du document ou de l'ensemble de données indexé : celle d'un individu ou d'une conversation créatrice. On pourra donc inventer plusieurs USL distincts pour catégoriser un document quelconque, selon les points de vue et les univers de discours. Rien n'empêche par ailleurs de réutiliser un USL pour lui enlever ou lui ajouter des propositions afin de personnaliser la catégorisation.

La stratégie générale de construction d'un USL consiste à décrire le "spectre sémantique" qui rend compte d'un document. Il ne s'agit surtout pas de chercher un équivalent du concept complexe à décrire en IEML dans *un seul terme* du dictionnaire. Plus le spectre sémantique est vaste et précis, c'est-à-dire exprimé par *plusieurs* unités de sens de *plusieurs* couches, et mieux fonctionneront les opérations de sélection (différence, inclusion, union...), les connexions avec d'autres USL et les calculs de distances sémantiques.

Un USL s'étage en couches. Les couches les plus basses désignent les composantes sémantiques les plus générales, tandis que les couches les plus hautes représentent les composantes sémantiques les plus spécifiques ou les plus pré-

cises.

3.2.5.2 L'exemple de "Wikipedia"

Convention d'écriture

Dans l'écriture des USL, les propositions distinctes sont toujours séparées par une barre oblique. Pour simplifier la lecture, j'ai séparé les *catsets* de couche L0, L1, L2, L3 par une barre oblique précédée et suivie d'un passage à la ligne. Les *propositions* de couche L4 et L5 sont également séparées par une barre oblique précédée et suivie d'un passage à la ligne. Les clauses distinctes de la même phrase sont réunies par un signe + précédé et suivi par un passage à la ligne. J'utiliserai les mêmes conventions dans les exemples suivants. L'USL ci-dessous est un exemple de texte IEML décrivant *Wikipedia*.

```
*
(U:+S:)
/
t. / d.
/
wo.y.- / wa.k.- / e.y.- / s.y.- / k.h.-
/
a.u.-we.h.-' / n.-y.-s.y.-'
/
(n.-y.-s.y.-' s.o.-k.o.-' E:E:A:.-',
+
n.-y.-s.y.-' b.i.-b.i.-' E:A:.m.-',)
/
u.e.-we.h.-' m.a.-n.a.-f.o.-' E:E:S:.-',
/
(e.-' (b.a.-b.a.-f.o.-'+ t.a.-b.a.-f.o.-') E:E:S:.-',_
+
e.-' s.e.-k.u.-' E:E:T:.-',_
+
(b.a.-b.a.-f.o.-'+ t.a.-b.a.-f.o.-') (E:F:.wa.-' + E:A:T:.-') E:E:U:.-',_
+
(b.a.-b.a.-f.o.-' + t.a.-b.a.-f.o.-') l.o.-m.o.-s.u.-' E:E:U:.-',_
+
s.e.-k.u.-'E:.-'E:B:.s.-', k.i.-b.i.-t.u.-', E:T:.x.-',_
+
s.e.-k.u.-', b.-u.-'E:.-'E:F:.wa.-', E:T:.x.-',_
+
k.i.-b.i.-t.u.-' b.x.-' E:E:U:.-',_)
**
```

Traduction de l'USL, dans l'ordre du script

L0 réseaux de connaissances

L1 mémoire / vérité

L2 s'orienter dans la connaissance / agir en vue du bien commun / synthétiser / connaissance organisée / création collective

L3 ouvrir l'espace public / encyclopédie

L4 encyclopédie de l'intelligence collective dans le cyberespace / volontaires produisant du matériel didactique / autorise une écriture et une édition plurielle et ouverte dans plusieurs langues, par le moyen d'un environnement logiciel collaboratif.

Tous les catsets de couche L0 à L3 sont composés de termes du dictionnaire dont le descripteur français est reproduit dans la traduction ci-dessus. Expliquons maintenant la traduction des trois propositions du catset de couche L4.

Première proposition de couche L4
*(n.-y.-s.y.-' s.o.-k.o.-' E:E:A:.-',
+
n.-y.-s.y.-' b.i.-b.i.-' E:A:.m.-',)**

Termes

n.-y.-s.y.-' encyclopédie

s.o.-k.o.-' intelligence collective

b.i.-b.i.-' médium numérique (ou cyberespace)

E:E:A:. génitif (complément du nom)

E:A:.m.- dans, au milieu de

Cette proposition est une phrase composée de deux clauses de couche L4 en relation d'union. Les clauses sont nominales puisque les deux séquences commencent par une consonne. Dans les deux clauses, le mode auxiliaire de couche 3 indique la relation entre l'unité substance de couche L3 et l'unité attribut de couche L3.

1. La première clause signifie : "encyclopédie de l'intelligence collective"
2. La seconde clause signifie : "encyclopédie dans le médium numérique"

On obtient donc (automatiquement) le diagramme de la figure 3.9 qui se traduit en français par "encyclopédie de l'intelligence collective dans le médium numérique". Dans ce diagramme, comme dans les précédents, les verbes et noms sont sont sur un fond gris et les auxiliaires sur un fond blanc.

Deuxième proposition de couche L4
*u.e.-we.h.-' m.a.-n.a.-f.o.-' E:E:S:.-',**

Termes

u.e.-we.h.-' produire du matériel didactique

m.a.-n.a.-f.o.-' volontaire

E:E:S:.- nominatif (marquant la relation entre verbe et sujet)

On remarquera qu'il n'y a ici aucune indication de nombre. Le fait qu'il n'y ait pas de pluriel n'indique pas qu'il s'agisse de singulier mais plutôt que l'on veut désigner un concept en général. De la même manière, l'absence de temps, de mode et de personne pour modifier le verbe indique qu'il faut comprendre

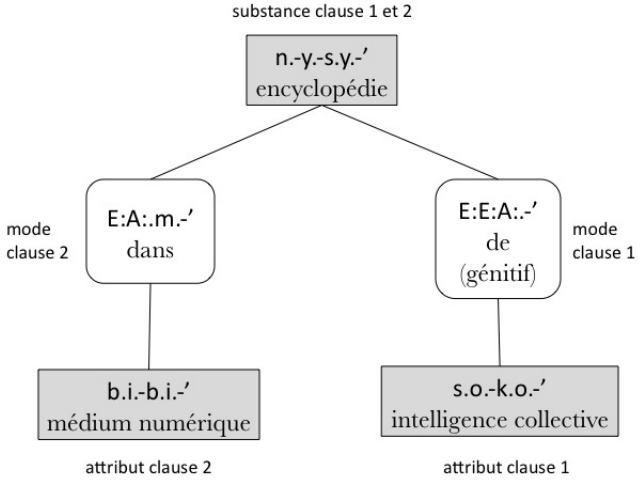

FIGURE 3.9 – Encyclopédie de l'intelligence collective dans le médium numérique

l'action de produire du matériel didactique en général. La signification de cette phrase verbale composée d'une seule clause est donc "volontaires produisant du matériel didactique". Beaucoup de langues naturelles obligent à choisir entre singulier et pluriel, ou à utiliser un article défini ou indéfini, ce qui n'est pas le cas d'IEML.

Proposition de couche L5

Termes

E:E:U:. attribut du nom

E:E:S:. nominatif

E:E:T:. complément d'objet direct

E:A:T:. beaucoup, très

e. pouvoir

E:B:.s.- dans un espace ouvert

E:T:.x.- auxiliaire de moyen

b.x.- environnement collaboratif

E:F:.wa.- auxiliaire marquant le pluriel

s.e.-k.u.-' compétence en lecture et écriture

b.-u.-' langue naturelle

b.a.-b.a.-f.o.-' auteur

t.a.-b.a.-f.o.-' éditeur

k.i.-b.i.-t.u.-' logiciel

l.o.-m.o.-s.u.-' libre

*

(e.-' (b.a.-b.a.-f.o.-'+ t.a.-b.a.-f.o.-') E:E:S:.-',_
+
e.-' s.e.-k.u.-' E:E:T:.-',_
+
(b.a.-b.a.-f.o.-'+ t.a.-b.a.-f.o.-') (E:F:.wa.-' + E:A:T:.-') E:E:U:.-',_
+
(b.a.-b.a.-f.o.-' + t.a.-b.a.-f.o.-') l.o.-m.o.-s.u.-' E:E:U:.-',_
+
s.e.-k.u.-'E:.-'E:B:.s.-', k.i.-b.i.-t.u.-', E:T:.x.-',_
+
s.e.-k.u.-', b.-u.-'E:.-'E:F:.wa.-', E:T:.x.-',_
+
k.i.-b.i.-t.u.-' b.x.-' E:E:U:.-',_)

**

La phrase IEML qui précède est à la couche L5 et elle est composée de sept clauses de couche L5 en relation d'union.

1. La première clause signifie que le sujet du "pouvoir" sont les lecteurs et les éditeurs.
2. La seconde clause signifie que l'objet du verbe "pouvoir" est la compétence de lecture-écriture.
3. La troisième clause insiste sur la multiplicité des lecteurs et des éditeurs.
4. La quatrième clause affirme la liberté de ces mêmes lecteurs et éditeurs.
5. La cinquième clause indique que la lecture-écriture "ouverte" se fait au moyen d'un logiciel.
6. La sixième clause indique que la compétence en lecture-écriture s'exerce au moyen de plusieurs langues.
7. La septième clause affirme que le logiciel a la propriété d'être un environnement collaboratif.

Ecrire ou lire cette phrase en entier revient à concevoir le réseau sémantique de la figure 3.10. Pour simplifier la représentation visuelle, le texte IEML n'y figure pas. Les noms et les verbes sont sur un fond gris tandis que les auxiliaires sont sur un fond blanc. Cette figure 3.10 donne une idée de ce à quoi pourrait ressembler le réseau sémantique en langue naturelle produit automatiquement à partir d'une phrase en IEML (la langue étant choisie par l'utilisateur).

3.2.5.3 Exemple d'opérations ensemblistes sur des USL

Il est possible d'effectuer des opérations ensemblistes sur des USL. Imaginons deux USL: "XML" et "Web_des_données"

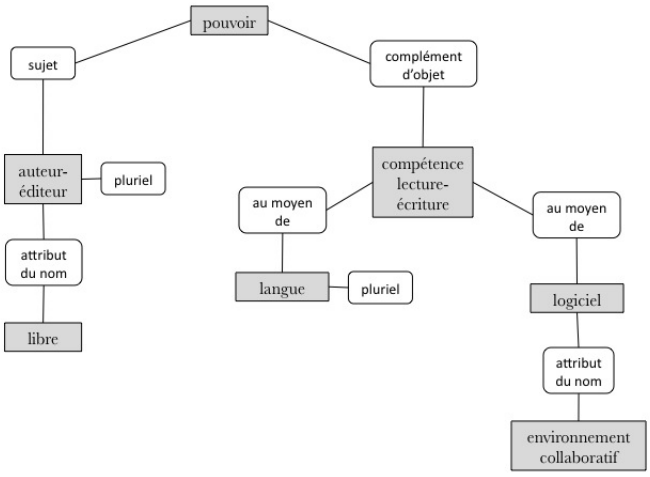

FIGURE 3.10 – Réseau syntagmatique d'une phrase IEML à sept clauses

"XML" *
(A:+S:)
/
b.
/
we.b.- / we.g.-
/
e.o.-we.h.-' / b.i.-b.i.-' / t.e.-d.u.-'
/
(i.i.-we.h.-', b.-j.-'E:.-'E:A:.g.-', E:T:.x.-',_
+
i.i.-we.h.-' s.a.-t.a.-' E:E:T:.-',_)
**

L0 réseaux de documents

L1 langage

L2 cultiver des systèmes d'information / unifier la documentation

L3 établir des normes et standards / médium numérique / répondre à des besoins d'information

L5 garantir la compatibilité de données grâce à la même structure formelle.

Tous les catsets de couche 0 à 3 sont composés de termes du dictionnaire dont le descripteur français est reproduit littéralement dans la traduction ci-dessus (les descripteurs apparaissent dans la traduction exactement dans l'ordre des

propositions IEML correspondantes). Expliquons maintenant la traduction de la catégorie de couche 5.

Termes

i.i.-we.h.-' assurer la compatibilité

b.-j.-' structure formelle

s.a.-t.a.-' données

E:E:T:. accusatif (complément d'objet direct)

E:A:.g.- comme, même

E:T:.x.- par le moyen de

1. La première clause verbale connecte le verbe en substance *i.i.-we.h.-',** "assurer la compatibilité" au nom en attribut *b.-j.-'E:.-'E:A:.g.-',** par l'auxiliaire en rôle de mode *E:T:.x.-,** marquant le moyen.
   *b.-j.-'E:.-'E:A:.g.-',** est lui-même un nom composé de deux morphèmes-termes :
   - le terme nominal *b.-j.-'**, qui signifie "structure formelle"
   - le terme auxiliaire *E:A:.g.-** modifiant le nom et qui signifie "même, comme".
   *b.-j.-'E:.-'E:A:.g.-',** signifie donc "même structure formelle"
2. La seconde clause verbale signifie : assurer la compatibilité (le verbe en substance *i.i.-we.h.-'**) des données (le nom en attribut *s.a.-t.a.-'**). La relation entre le verbe en substance et le nom en attribut est décrite par le mot auxiliaire en rôle de mode *E:E:T:.-'** marquant le complément d'objet direct.

La phrase complète donne "Garantir la compatibilité de données grâce à (par le moyen de) la même structure formelle."

On notera que les termes *s.a.-t.a.-'** (données) et *s.a.-t.a.-f.o.-'** (informaticien) ont la même substance et le même attribut, *s.a.-t.a.-f.o.-'** étant le/la spécialiste de l'organisation et des manipulations de *s.a.-t.a.-'**. Il en est de même pour *b.-j.-'** (structure formelle) et *b.-j.-s.y.-'** (mathématiques), l'un étant l'objet de l'autre.

Donnons maintenant l'exemple d'un second USL, que qui sera ensuite l'objet d'une union et d'une intersection avec le précédent.

"Web_des_données" *

(U:+S:) / (A:+S:)
/
d.
/ s.x.- / x.j.-
/
e.o.-we.h.-' / b.i.-b.i.-'
/
e.o.-we.h.-' b.i.-b.i.-' E:E:B:.-',
/

s.a.-t.a.-’ b.i.-l.i.-t.u.-’ E:E:U:.-’,
/
(t.e.-t.u.-wa.e.-’ k.i.-b.i.-t.u.-’ E:E:A:.-’,
+
k.i.-b.i.-t.u.-’ b.e.-’ E:A:.x.-’,)
**

L0 réseaux de connaissance / réseaux de documents

L1 vérité

L2 projet technique / évolution du calcul

L3 établir des normes et standards / médium numérique

L4 établir des normes et standards pour le médium numérique / données liées (*linked data*) / conception de robots logiciels avec des capacités de raisonnement

La traduction n'est explicitée que pour les deux dernières propositions de couche L4, en commençant par donner la traduction des termes.

Termes

E:E:U:. attribut du nom

E:E:A:. génitif

E:E:B:. datif (au bénéfice de)

E:A:.x.- avec

b.e.- capacité de raisonnement

s.a.-t.a.-’ données

b.i.-l.i.-t.u.-’ relié

t.e.-t.u.-wa.e.-’ compétence conception de logiciels

k.i.-b.i.-t.u.-’ logiciel

1. La seconde proposition de couche L4 est une phrase nominale composée d'une seule clause : les données *s.a.-t.a.-’** sont (attribut du nom *E:E:U:.**) liées *b.i.-l.i.-t.u.-’**, ce qui traduit le concept de *“linked data”*.
2. La troisième proposition de couche L4 est une phrase à deux clauses. La première clause indique une compétence en conception *t.e.-t.u.-wa.e.-’** de (*E:E:A:.** génitif) logiciel *k.i.-b.i.-t.u.-’** et la seconde clause indique que le logiciel a une capacité de raisonnement (avec : *E:A:.x.-**, capacité de raisonnement : *b.e.-**).

Résultats de deux opérations sur les USL précédents Les opérations ensemblistes sur les USL sont toujours effectuées sur des ensembles de propositions (et donc sur des ensembles de séquences) *couche par couche*.

L'*intersection* des deux USL précédents donne un nouvel USL (que l'on pourrait appeler “Internet_standards”) contenant une proposition de couche 0 et deux propositions de couche 3.

*
(A:+S:)
/
e.o.-we.h.-’ / b.i.-b.i.-’
**

L0 réseaux de documents

L3 établir des normes et standards / médium numérique

L’*union* des deux USL donne l’USL que l’on pourrait appeler “Grand_Web_des_données”

*
(U:+S:) / (A:+S:)
/
b. / d.
/
we.g.- / we.b.- / s.x.- / x.j.-
/
e.o.-we.h.-’ / b.i.-b.i.-’ / t.e.-d.u.-’
/
e.o.-we.h.-’ b.i.-b.i.-’ E:E:B:.-’,
/
s.a.-t.a.-’ b.i.-l.i.-t.u.-’ E:E:U:.-’,
/
(t.e.-t.u.-wa.e.-’ k.i.-b.i.-t.u.-’ E:E:A:.-’,
+
k.i.-b.i.-t.u.-’ b.e.-’ E:A:.x.-’,)
/
(i.i.-we.h.-’, b.-j.-’E:.-’E:A:.g.-’, E:T:.x.-’,_
+
i.i.-we.h.-’ s.a.-t.a.-’ E:E:T:.-’,_)
**

L0: réseaux de connaissance / réseaux de documents

L1: langage / vérité

L2: unifier la documentation / cultiver des systèmes d’information / projet technique / évolution du calcul

L3: établir des normes et standards / médium numérique / répondre à des besoins d’information

L4: établir normes et standards pour le médium numérique / données liées (*linked data*) / conception de robots logiciels avec des capacités de raisonnement

L5: garantir la compatibilité de données grâce à une même structure formelle

CANAUX	**Multiplication** $\otimes$	**Addition** $\oplus$
Ordre $>$	- mot $\otimes >$ morphème - clause $\otimes >$ mot - clause de phrases $\otimes >$ phrase - clause hyp. $\otimes >$ texte	- morphème $\oplus >$ terme - phrase $\oplus >$ clause - texte $\oplus >$ proposition - hypertexte $\oplus >$ clause hyp.
Symétrie $\sqcap$	- mot $\otimes\sqcap$ mot inversé - clause $\otimes\sqcap$ clause inversée - cl. de ph. $\otimes\sqcap$ cl. de ph. inversée - clause hyp. $\otimes\sqcap$ cl. hyp. inversée	- terme $\oplus\sqcap$ terme - mot $\oplus\sqcap$ mot - clause $\oplus\sqcap$ clause - proposition $\oplus\sqcap$ proposition - clause hyp. $\oplus\sqcap$ clause hyp.

TABLE 3.5 – Les 4 types de canaux des circuits énonciatifs

3.2.6 Représentation des relations syntaxiques entre unités d'énonciation

Il existe en IEML deux grands types de circuits sémantiques, les circuits paradigmatiques, que nous avons étudié au chapitre 3.1 et les circuits énonciatifs, dont nous venons de voir les principaux aspects linguistiques dans les sections qui précèdent. Rappelons ici ce que nous savons déjà des circuits énonciatifs : ils sont constitués par une hiérarchie complexe de propositions, de textes (additions de propositions) et d'hypertextes (additions et multiplications de textes). Les propositions sont elles-mêmes construites à partir d'une hiérarchie complexe de morphèmes, de mots, de clauses, de phrases et de phrases de phrases. A la base de cette hiérarchie d'unités de sens se trouvent les termes, qui sont définis par les clés du dictionnaire (voir la section 3.2.1.2).

Il existe en IEML un strict parallélisme entre les relations syntaxiques qui connectent les unités de sens, d'une part, et les canaux qui connectent les réservoirs des circuits énonciatifs, d'autre part. Ce parallélisme est médié par les opérations d'addition et de multiplication. On se souvient que ces opérations commandent des relations d'ordre et de symétrie entre leurs variables de manière assez simple :

1. pour l'addition, une relation d'ordre relie chacune des variables d'entrée à la variable de sortie et une relation de symétrie relie les variables d'entrée de la même opération ;
2. pour la multiplication, une relation d'ordre relie chacune des trois variables d'entrée à la variable de sortie et une relation de symétrie relie les variables de sortie dont deux variables d'entrée ont échangé leurs rôles.

Le tableau 3.5 passe en revue la représentation des relations syntaxiques entre unités de sens par des canaux entre réservoirs. Commentons brièvement les quatre types de canaux présentés dans ce tableau.

3.2.6.1 Canaux d'ordre multiplicatif

Les canaux d'ordre multiplicatifs connectent des unités de sens de niveau n à l'unité de sens *de niveau*[10] *n+1* qui les comprend. Par exemple, des morphèmes et le mot qui les contient, des mots et la clause qui les articule, et ainsi de suite. Je rappelle ici qu'un texte (USL) et les propositions qu'il contient *ne sont pas* connectés par des canaux d'ordre multiplicatifs mais par des canaux d'ordre additifs.

3.2.6.2 Canaux d'ordre additif

Les canaux d'ordre additifs connectent des unités de sens de niveau n à une unité de sens *de même niveau*[11] *n* qui les contient ou les englobe. Par exemple : des termes et le morphème qui les contient, des clauses et la phrase qui les contient, etc.

3.2.6.3 Canaux de symétrie multiplicative

Les canaux de symétrie multiplicative connectent les unités de même niveau dont deux sèmes inversent leurs rôles et dont le troisième sème est identique. Par exemple, la phrase "L'histoire du moteur est un agent du moteur de l'histoire" se traduit ainsi en IEML :

*k.o.-t.o.-'E:.-'E:A:.p.-',n.i.-s.i.-'E:.-'E:A:.p.-',E:E:A:.-',_n.i.-s.i.-'E:.-'E:A:.p.-',k.o.-t.o.-'E:.-'E:A:.p.-',E:E:A:.-',E:B:.x.-', _;**

Dans ce cas, les deux clauses

*k.o.-t.o.-'E:.-'E:A:.p.-',n.i.-s.i.-'E:.-'E:A:.p.-',E:E:A:.-',_**

et

*n.i.-s.i.-'E:.-'E:A:.p.-',k.o.-t.o.-'E:.-'E:A:.p.-',E:E:A:.-',**

sont en relation de symétrie multiplicative.

Termes

E:E:A:. génitif

E:A:.p.- article défini

E:B:.x.- agent, cause efficiente

k.o.-t.o.-' histoire

n.i.-s.i.-' moteur

Au niveau inter-textuel, on trouvera des canaux de symétrie multiplicative entre des USL qui s'entre-citent.

3.2.6.4 Canaux de symétrie additive

Les canaux de symétrie additive interconnectent en graphe complet les unités de niveau n qui concourent à la construction d'une unité englobante de même niveau n. Par exemple, les termes d'un même morphème, les clauses de la même phrase, les propositions du même texte, etc.

10. Par niveau j'entends ici la couche, le degré, l'ordre, etc.
11. Voir la note précédente.

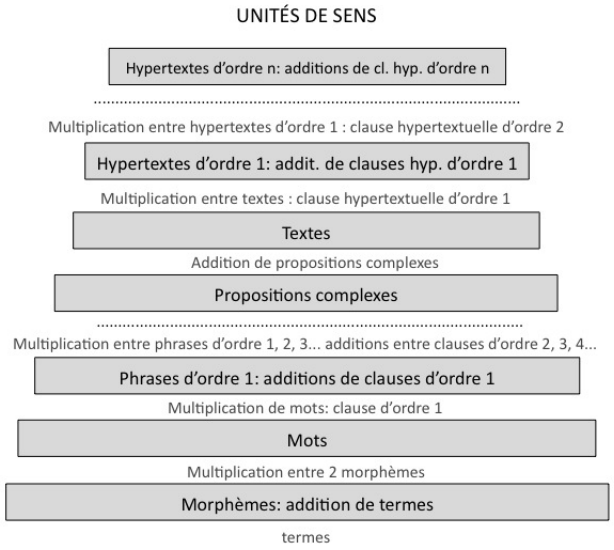

FIGURE 3.11 – La hiérarchie enchevêtrée des unités de sens

3.2.7 Résumé de la hiérarchie des unités de sens

La figure 3.11 résume la hiérarchie des unités de sens en IEML. Cette hiérarchie est "enchevêtrée". En effet, d'un côté, le sens des unités supérieures est conditionné par le sens des unités inférieures mais, d'un autre côté, le sens des unités inférieures est réciproquement conditionné par celui des unités supérieures, qui forment, degré par degré, leurs contextes respectifs.

3.2.8 Constructions correctes et incorrectes***

3.2.8.1 Codage

Afin de représenter de manière concise les patterns grammaticaux selon lesquels sont arrangés les unités de sens, Ils seront codés de la manière suivante :

V représente une séquence vide

Z représente un terme

R représente un morphème

W' représente un mot morphème

W" représente un mot fléchi

W représente un mot morphème ou bien (*ou exclusif*) un mot fléchi

$\mathbf{C}^n$ représente une clause d'ordre 1, ou 2, ou 3, etc. n indiquant le numéro d'ordre de la clause

C représente une clause, quelque soit son numéro d'ordre

P^n représente une phrase d'ordre 1, ou 2, ou 3, etc. n indiquant le numéro d'ordre de la phrase

P représente une phrase, quelque soit son numéro d'ordre

Q représente une unité de sens quelconque (non entièrement vide).

3.2.8.2 Promotion

Dans toute addition $\oplus$ ou multiplication $\otimes$ d'unités de sens Q = {Z, R, W, C, P} les variables aux couches les plus basses sont promues par une séquence vide jusqu'à ce que soit atteinte la couche de la variable qui est à la couche la plus haute. Au sujet de la promotion, renvoyons à la section 2.2.6.3 pour l'aspect syntaxique et à la section 3.2.1.4 pour l'aspect sémantique.

3.2.8.3 Constructions grammaticales

— R = (Z $\oplus$ Z $\oplus$ ...)
— W' = R
— W" = (R $\otimes$ E $\otimes$ R)
— C^1= {(W $\otimes$ W $\otimes$ W), (W' $\otimes$ E $\otimes$ W"), (W" $\otimes$ E $\otimes$ W'), (W" $\otimes$ E $\otimes$ W"), (W $\otimes$ E $\otimes$ E)}
Note 1 : Une construction du type (W' $\otimes$ E $\otimes$ W') se ramène à (R $\otimes$ E $\otimes$ R), qui est forcément un mot. Ce ne peut donc être une clause.
Note 2 : les clauses du type (W $\otimes$ E $\otimes$ E) ne peuvent être utilisées que dans les phrases d'ordre 1 P^1
— P^1 = (C^1 $\oplus$ C^1 $\oplus$ ...)
— C^2 = {(P^1 $\otimes$ P^1 $\otimes$ P^1), (P^1 $\otimes$ E $\otimes$ P^1), (P^1 $\otimes$ E $\otimes$ E)}
Note : les clauses du type (P^1 $\otimes$ E $\otimes$ E) ne peuvent être utilisé que dans les phrase d'ordre 2 P^2.
— P^2 = (C^2 $\oplus$ C^2 $\oplus$ ...)
— Pour les clauses d'ordre supérieur à 2 :
C^{n+1} = {(P^n $\otimes$ P^n $\otimes$ P^n), (P^n $\otimes$ E $\otimes$ P^n), (P^n $\otimes$ E $\otimes$ E)}
Note : les clauses du type (P^n $\otimes$ E $\otimes$ E) ne peuvent être utilisées que dans les phrases d'ordre n+1 P^{n+1}.
— Pour les phrases d'ordre supérieur à 2 :
P^n = (C^n $\oplus$ C^n $\oplus$ ...)

3.2.8.4 Opérations non-grammaticales

— L'opération de forme (Q $\otimes$ Q $\otimes$ E) est non-grammaticale, parce que la relation entre substance et attribut resterait non-définie.
— L'opération de forme (E $\otimes$ E $\otimes$ Q) est non-grammaticale puisque le mode n'aurait rien à modifier ou à connecter.
— L'opération de forme (E $\otimes$ E $\otimes$ E), est non grammaticale, parce qu'elle ne signifie rien, sauf comme série vide suivant une série non complètement vide (voir plus haut en 3.2.8.2)

— Les opérations de la forme (W $\oplus$ W $\oplus$ ...) et (P $\oplus$ P $\oplus$ ...) sont non-grammaticales. Il faut employer à la place des opérations de forme (C $\oplus$ C $\oplus$ ...).

3.2.8.5 L'attribut vide dans les opérations de forme (Q $\otimes$ E $\otimes$ Q)

L'attribut est *toujours* occupé par une série vide dans les mots fléchis et il peut être occupé par une série vide dans les clauses. Dans ces deux cas, le mode précise le sens de la substance au lieu de définir la relation entre la substance et l'attribut.

3.3 Principes de l'algorithme de transcodage du Rhizome vers la Sphère sémantique***

3.3.1 Les circuits énonciatifs ont la même structure que les circuits syntagmatiques

Si nous disposons d'un algorithme de construction des circuits syntagmatiques (qui correspondent aux propositions), il est évident que nous pouvons construire un algorithme de construction des circuits énonciatifs en général (qui comprennent en plus les textes et les hypertextes). En effet, premièrement, les propositions d'un même *texte* sont simplement des réservoirs interconnectés par des canaux de symétrie additive et qui convergent par des canaux d'ordre additif vers le réservoir de l'USL qui les contient. Signalons cependant que l'auteur d'un texte peut supprimer des canaux de symétrie additive entre propositions (c'est-à-dire réduire la redondance du circuit) à condition que le circuit final entre les propositions soit connexe. Deuxièmement, les *hypertextes* relèvent exactement de la même structure enchevêtrée d'additions et de multiplications que les syntagmes. Contentons nous donc de montrer ici que nous disposons effectivement d'un algorithme de construction des circuits syntagmatiques.

3.3.2 La base de l'algorithme de construction des circuits sémantiques est l'algorithme de transcodage du Script vers le Rhizome

Pour obtenir un algorithme de construction des circuits syntagmatiques, il suffit d'utiliser l'algorithme de transcodage Script vers Rhizome décrit en 2.3 avec les modifications qui suivent.

3.3.2.1 Changement de nature des unités manipulées

Au lieu de bulbes et de filaments, on parlera de réservoirs représentant des unités de sens et de canaux représentant des relations grammaticales entre unités de sens.

3.3.2.2 Arrêt de la lecture aux termes

La lecture s'arête de descendre vers les feuilles des arbres d'ordre additif et multiplicatif *lorsque elle arrive à un terme*, ce qui implique d'ajouter un "test de terme" à l'algorithme.

3.3.3 Qualification des unités de sens

Une fois complétée l'architecture du circuit selon la méthode qui vient d'être décrite, l'algorithme d'énonciation doit remonter les canaux d'ordre multiplicatif et additif à partir des termes vers la proposition complète pour *qualifier les différentes unités de sens* (morphèmes, mots, clauses de type nominal, verbal ou auxiliaire). Tout ceci se fait selon les règles décrites en 3.2.8.

3.3.4 Visualisation des réseaux sémantiques

3.3.4.1 Visualisation du réseau sémantique d'une proposition

Une fois qualifiées toutes les unités de sens de la proposition, on peut ensuite visualiser automatiquement son réseau sémantique pour le lecteur humain. Il suffit pour cela de représenter...

- chaque terme IEML par son équivalent-dictionnaire dans la langue naturelle choisie,
- chaque morphème comme un graphe complet des termes qu'il contient,
- chaque mot fléchi (W") comme une connexion entre un morphème racine (en rôle de substance) et un morphème de flexion (en rôle de mode).
- chaque clause (à son degré respectif) comme une connexion entre l'unité en rôle de substance et l'unité en rôle d'attribut, cette connexion étant étiquetée par l'unité en rôle de mode,
- chaque phrase comme l'union des connexions représentées par ses clauses en relation d'addition.

Nous avons déjà examiné plus haut des exemples de ce type de représentation. Renvoyons notamment aux figures 3.10, 3.9, 3.8 et 3.7) dans lesquelles les unités en rôle de substance et d'attribut sont représentés en gris et les unités en rôle de mode en blanc.

3.3.4.2 Visualisation des textes et hypertextes

Un texte (USL) sera représenté par défaut comme un graphe complet entre les propositions qu'il contient. Quant aux hypertextes (de degrés successifs), ils se construisent exactement selon le même principe que les phrases, sauf que les clauses hypertextuelles de premier ordre sont construites à partir de textes au lieu d'être construites à partir de mots. Les textes en rôle de mode sont les étiquettes ou « notes » du lien du texte substance vers le texte attribut.

Chapitre 4

Transcodage syntaxe-sémantique

« L'algorithme est l'idiome de la science moderne » Bernard Chazelle

4.1 Introduction

Ce chapitre présente les algorithmes de transcodage d'IEML. Le transcodage de la syntaxe en sémantique peut se décomposer en trois transformations : la transformation d'IEML algébrique en Script IEML dans la section 4.2, du Script IEML vers le Rhizome IEML dans la section 4.3 et finalement du Rhizome IEML vers la Sphère sémantique dans la section 4.4.

Certains algorithmes utilisent des variables globales et des fonctions qui sont décrites respectivement dans les sections 4.1.1 et 4.1.2.

4.1.1 Variables globales

1. Le paramètre `MinLayer` contrôle la profondeur d'analyse. Son utilisation garantit que, pour une expression s en Script à la couche L_n, toutes les couches l de s telles que $MinLayer \leq l \leq L_n$ sont analysées. Utilisé dans l'algorithme 5.
2. Le paramètre `MultSym` décide de la création de relations de symétrie multiplicative. Utilisé dans l'algorithme 5.
3. Le paramètre `MultOrd` contrôle les relations d'ordre multiplicatif qui sont créés. Ce paramètre définit trois variables logiques, `substance`, `attribute` et `mode`. Des relations sont créées si ces variables logiques ont la valeur *true*. Voir la section 4.1.2.
4. Le paramètre `RestrictedScript` est une liste d'expressions en Script. Les membres de cette liste sont exclus de l'analyse. Utilisé dans l'algorithme 5.

5. Le paramètre `Terms` est une liste d'expressions en Script. Les membres de cette liste sont utilisés à différentes fins. Utilisé dans les algorithmes 5 et 6.
6. La structure de données `Chars` enregistre une expression Script s de manière à permettre l'itération de remplacements de caractères de couche zéro, la modification de ces caractères, et la modification d'une expression en script à partir de ces caractères. Utilisé dans l'algorithme 5.
7. Le paramètre $\mathtt{E}^{\mathtt{A}}$ est une liste de relations d'ordre additif. Utilisé dans les algorithmes 8 et 10.
8. Le paramètre $\mathtt{E}^{\mathtt{B}}$ est une liste de relations de symétrie additive. Utilisé dans les algorithmes 8 et 10.
9. Le paramètre $\mathtt{E}^{\mathtt{C}}$ est une liste de relations d'ordre multiplicatif. Utilisé dans les algorithmes 8, 9 et 10.
10. Le paramètre $\mathtt{E}^{\mathtt{D}}$ est une liste de relations multiplicatives. Utilisé dans les algorithmes 8, 9 et 10.
11. Le paramètre $G = (V, E)$ est un graphe définissant un rhizome où $E = E^A \cup E^B \cup E^C \cup E^D$. Utilisé dans les algorithmes 5, 6, 7, 8, 9, 10 et 11.
12. Les paramètres W' et W'' sont des listes d'expression en Script (mots). Utilisé dans les algorithmes 8, 9 et 12.
13. Le paramètre $\mathtt{C}^{\mathtt{n}}$ est une liste d'expressions en Script de couche n (clauses). Utilisé dans les algorithmes 7, 9, 10, 12.
14. Le paramètre $\mathtt{P}^{\mathtt{n}}$ est une liste d'expressions en Script de couche n (phrases). Utilisé dans l'algorithme 7, 10 et 12.
15. Le paramètre $\mathtt{H}_{\mathtt{NAT}}$ est une table de hachage contenant une correspondance entre des expressions en Script et des expressions en langue naturelle. Utilisé dans l'algorithme 12.
16. Les paramètres $\mathtt{G}_{\mathtt{SEM}}$, $\mathtt{G}^{-}_{\mathtt{SEM}}$ et $\mathtt{G}^{=}_{\mathtt{SEM}}$ sont des graphes. Utilisés dans l'algorithme 12.

4.1.2 Fonctions globales

1. `IEMLOrder`(`S`). Implémente les règles de rangement de la section 2.2.7 et donne comme résultat une liste d'expressions Script ordonnée selon le nombre de multiplications dans cette expression. Les expressions avec le plus petit nombre de multiplications sont rangées en premier dans cette liste. L'ensemble `S` contient des ensembles d'expressions algébriques IEML. Utilisé dans l'algorithme 1.
2. `IEMLSeme`(`s`, `i`). Trouve le $i$ème sème de s, où s est une expression algébrique IEML de couche $L > 0$ et i est un entier $1 \leq i \leq 3$. Utilisé dans l'algorithme 3.
3. `IEMLReplace`(`S`). Implémente les règles de remplacement définies à la section 2.2.6.3. L'ensemble `S` contient des ensembles d'expressions algébriques IEML. Utilisé dans l'algorithme 4.

4. `Layer(script)`. Trouve la plus haute couche de l'expression en Script qui est analysée, la couche l étant comprise entre 0 et 6 ($0 \leq l \leq 6$). Utilisé dans l'algorithme 5.
5. `Parse(s, construct)`. Trouve la configuration (*construct*) spécifiée dans le l'expression Script s en input, à la plus haute couche de cette expression. Trois types de configurations sont définies, *addition* ($\oplus$), *multiplication* ($\otimes$) et *caractère* (©). La configuration *addition* dirige la fonction vers la découverte des sous-scripts qui sont à la même couche que s et donne comme résultat un ou plus de ces sous-scripts. La configuration *multiplication* dirige la fonction vers la découverte des trois sèmes de s et donne comme résultat ces trois sèmes. La configuration *caractère* dirige la fonction vers la découverte des caractères de couche 0 de s et donne comme résultat ces caractères. Utilisé dans les algorithmes 5 et 6.
6. `RoleInterversion(s)`. Calcule les permutations de sèmes du script s fourni en input et renvoie une expression en Script. Utilisé dans l'algorithme 5.
7. `Expand(c)`. Calcule les composantes sous-jacentes du caractère c. L'input doit être à la couche 0. Par exemple, le caractère (U:+B:) est décomposé pour donner l'ensemble {U:, B:}, le caractère (I:) est décomposé dans l'ensemble {E:, F:} et le caractère (F:) est décomposé en un ensemble {O:, M:}. Le résultat peut aussi être un ensemble vide. Utilisé dans l'algorithme 6.
8. `CreateAlgebraic(Chars)`. Crée une expression algébrique IEML à partir de la structure *Chars*. Pour les détails de cette structure, voir la section 4.1.1, en particulier le point 6. Utilisé dans l'algorithme 6.
9. `Pattern(template)`. Chaque expression Script peut être caractérisée par ses sèmes. Nous ne nous intéressons ici qu'aux sèmes et sous-sèmes qui représentent une séquence vide. Par exemple, une valeur de pattern (*template*) de `SES` oblige la fonction à vérifier si le second sème - et lui seul - d'une expression en Script est une séquence vide, alors qu'une valeur de pattern de $SES \otimes EEE \otimes SES$ oblige la fonction à vérifier si le premier et troisième sème d'une expression en Script correspondent au pattern `SES`, et si le second sème est une séquence vide. Utilisé dans les algorithmes 8, 9 et 10.
10. `DetType(s)`. Implémente les règles de rangement définies à la section 2.2.7 et classe l'expression s dans un des trois types *nominal*, *verbal* ou *auxiliary* (auxiliaire) tel que défini à la section 3.2.2. Utilisé dans l'algorithme 8.
11. `Clique(s)`. Trouve tous les sous-scripts de l'expression Script s qui sont en relation de symétrie additive et donne comme résultat le graphe qui connecte ces sous-scripts. Utilisé dans l'algorithme 12.
12. `Translate(s, hashtable)`. Etablit la correspondance entre une expression Script s donnée et une valeur fournie par une table de hachage. Renvoie la valeur en question. Utilisé dans l'algorithme 12.

4.1.3 Fonctions globales de création de relations

Il existe trois fonctions qui ont le même type d'inputs mais qui calculent différents types de relations. Les inputs sont une expression Script s (ou un ensemble contenant exactement une expression Script) et un ensemble S d'expressions en Script. Ces fonctions ont pour résultat un ensemble de sommets et d'arêtes.

Ces fonctions ont deux inputs, un script racine *(root)* s - ou un ensemble contenant exactement un script -, et un ensemble S de scripts. Pour les trois fonctions, les relations ne sont pas créées pour chaque $s_i \in S$ quand $s_i \in$ *RestrictedScript*. De plus, la fonction `AdditionRelations` crée deux types de relations, des *relations d'ordre additif* et *des relations de symétrie additive*. Ces fonctions sont :

1. `AdditionRelations`(`s`, `S`). Utilisé dans les algorithmes 5 et 6
2. `MultiplicationOrder`(`s`, `S`). Utilisé dans l'algorithme 5
3. `MultiplicationSymmetry`(`s`, `S`). Utilisé dans l'algorithme 5

Pour créer les relations appropriées, la fonction `MultiplicationOrder` dépend des réglages de `MultOrd`, et son ensemble d'input S doit être de trois éléments (les sèmes).

4.2 De l'Algèbre vers le Script

L'algorithme 4.1 trouve une expression en Script à partir d'une catégorie IEML. Par exemple, pour un ensemble d'input $\{USE, UBE, UTE, ASE, ABE, ATE\}$, l'algorithme 4.1 va donner *O:M:.** en résultat.

Algorithme 4.1 Algorithme SCRIPT_GENERATOR

input : (S) where S is a set of IEML strings
output : Script expression

```
1  K ;                                  // ensembles d'ensembles de séquences
2  begin
3      K ⟵ ∅ ;                          // initialise l'ensemble de candidats
4      K ⟵ SEME_COMPRESSOR(S, K) ;      // trouve les solutions candidates
5      return IEMLOrder(K)
```

Pour plus de clarté, l'algorithme 1 4.1 délègue délègue la plus grande partie de son calcul à l'algorithme 4.2 et sa complexité spatio-temporelle est donc proportionnelle à la complexité de l'algorithme 4.2.

4.2.1 L'algorithme SCRIPT_COMPRESSOR

Algorithm 4.2 Algorithme SCRIPT_COMPRESSOR

input : (S, K) where S is a set of IEML strings and K is a set of sets of IEML strings
output : Set of sets of IEML strings K

```
6  C, Q ;                                  // ensemble d'ensemble de séquences
7  begin
8      C ⟵ SEME_MATCHER(S) ;                        // combinaisons de sèmes
9      Q ⟵ SCRIPT_ SOLVER(C, S, ∅, ∅) ;    // trouve toutes les solutions
       pour S
10     if Q\{S} = ∅ then                          // l'input égale l'output
11         K ⟵+ S ;                        // ajoute les solutions candidates
12         return K ;                                  // termine l'analyse
13     for ∀q ∈ Q do                                 // continue l'analyse
14         SCRIPT_COMPRESSOR(q, K) ;                     // appel recursif
15     return K ;                                      // termine l'analyse
```

Le but de l'algorithme 4.2 est de s'assurer qu'aucune réécriture de caractère ne peut plus avoir lieu. Souvenons-nous qu'une séquence $\{U:+A:\}$ est réécrite comme $O:$, qu'une séquence $\{E:+F:\}$ est réécrite comme $I:$ et qu'une séquence $\{O:+M:\}$ est réécrite comme $F:$. Il existe une limite à la quantité de réécritures qui peuvent avoir lieu et donc la complexité de l'algorithme 4.2 est proportionnelle à $K \cdot C_{SEME\_MATCHER} \cdot C_{SCRIPT\_SOLVER}$, où K est un scalaire qui

rend compte des vérification des membres de la liste et $C_{SEME\_MATCHER}$ and C_{SCRIPT_SOLVER} représentent respectivement la complexité des algorithmes 4.3 et 4.4.

L'algorithme 4.2 est assez simple. Il comprend une condition terminale à la ligne 5 et un appel récursif à la ligne 9. L'algorithme termine quand l'input (S) est égal aux valeurs calculées à la ligne 4 (Q), ce qui signifie qu'il n'y a pas d'autres solutions potentielles à considérer.

4.2.2 L'algorithme SEME_MATCHER

Etant donné un ensemble d'input de séquences IEML de même couche, l'algorithme 4.3 trouve tous les groupes distincts de séquences qui partagent exactement deux sèmes. Par exemple, pour un ensemble d'input $\{USE, UBE, UTE, ASE, ABE, ATE\}$, l'algorithme 4.3 produira l'ensemble qui se trouve dans la table 4.1.

C
$\{USE, ASE\}$
$\{UBE, ABE\}$
$\{UTE, ATE\}$
$\{ASE, ABE, ATE\}$
$\{USE, UBE, UTE\}$

TABLE 4.1 – Résultat de l'algorithme 4.3 pour l'ensemble $\{USE, UBE, UTE, ASE, ABE, ATE\}$

Les lignes 1 et 2 listent les variables temporaires et leurs types. La ligne 4 accomplit l'initialisation, les lignes 5 à 19 représentent la principale boucle logique, les lignes 20 à 23 enregistrent le résultat de l'algorithme.

Chaque séquence est composée de trois sous-séquences de même longueur (sèmes) qui sont comparées et groupées par la boucle logique principale (lignes 9 à 19). Si un groupe particulier ne contient pas deux membres ou plus, ou s'il a déjà été trouvé, il n'est pas enregistré (ligne 21).

La complexité spatio-temporelle de l'algorithme 4.3 est respectivement proportionnelle à ln^2 et ln , où l est la longueur des séquences d'input, et n est la taille de l'ensemble d'input.

4.2.3 L'algorithme SCRIPT_SOLVER

L'algorithme 4.4 trouve toutes les expressions uniques résultant de la considération de tous les groupes distincts obtenus à partir de l'ensemble d'input S. Par exemple, pour l'ensemble d'input $\{USE, UBE, UTE, ASE, ABE, ATE\}$ et pour les groupes de la table 4.1, l'algorithme 4.4 produira l'ensemble de la table 4.2.

Les lignes 1 et 2 listent les variables temporaires et leur type. Les lignes 13 à 26 représentent la boucle logique principale. Les lignes 4 à 12 représentent

Algorithme 4.3 Algorithme SEME_MATCHER

input : Ensemble S de séquences IEML
output : Ensemble C d'ensembles de séquences IEML

```
16 a_i, b_i ;                                                  // séquence
17 D_i ;                                              // ensemble de séquences
18 begin
19     C ⟵ ∅ ;                                        // initialise C avec ∅
20     for all x ∈ S do
21         for i ∈ {1,2,3} do
22             D_i ⟵ x ;                              // initialise D_i avec x
23             a_i ⟵ IEMLSeme(x, i)
24         for all y ∈ S\x do
25             for i ∈ {1,2,3} do
26                 b_i ⟵ IEMLSeme(y, i)
27             if a_1 = b_1 then
28                 if a_2 = b_2 then
29                     D_3 ⟵+ y ;                      // ajoute y à D_3
30                 if a_3 = b_3 then
31                     D_2 ⟵+ y
32             if a_2 = b_2 then
33                 if a_3 = b_3 then
34                     D_1 ⟵+ y
35         for i ∈ {1,2,3} do
36             if ||D_i|| ≥ 2 and D_i ∉ R then
37                 C ⟵+ D_i ;                          // ajoute D_i à C
38     Return C
```

Q
$\{OTE, OSE, OBE\}$
$\{UME, AME\}$

TABLE 4.2 – Résultat de l'algorithme 4.4 pour l'ensemble $\{USE, UBE, UTE, ASE, ABE, ATE\}$ et l'input de la table 4.1

la terminaison logique de la récursion. Si l'ensemble C est vide, s'il contient seulement un élément, ou si l'intersection entre chacun de ses éléments est égale à un ensemble vide, alors l'algorithme se termine. Les termes non utilisés de S sont ajoutés à la solution potentielle (ligne 9), et si cette solution potentielle n'a pas été déjà trouvée, elle est ajoutée à l'ensemble des solutions possibles (ligne 11). Dans la boucle logique principale l'espace des solutions est réduit (lignes

ligne 42:
{∀c i , c j ∈C | c i ∩c j } = ∅ devrait être {∀c i , c j ∈C | c i ∩c j = ∅}
ligne 53:
{∀c i ∈C\{c} | c ∩c i } = ∅devrait être {∀c i ∈C\{c} | c ∩c i = ∅}

Algorithm 4.4 Algorithme SCRIPT_SOLVER

input : (C, S, R, Q) où C, R et Q sont des ensembles d'ensembles de séquences IEML, S est un ensemble de séquences IEML
output : Ensemble d'ensembles de séquences IEML Q

39 $\overline{S}$; // ensemble de séquences
40 $\overline{C}$, $\overline{R}$; // ensembles d'ensembles de séquences
41 **begin**
42 **if** $\{\forall c_i, c_j \in C \mid c_i \cap c_j\} = \emptyset$ **then** // inclut C = ∅
43 **for** $\forall c \in C$ **do**
44 $R \xleftarrow{+} c$; // ajoute le reste de C
45 $S \longleftarrow S \backslash c$; // enlève les séquences utilisées
46 **for** $\forall s \in S$ **do**
47 $R \xleftarrow{+} \{s\}$; // ajoute le reste de S
48 **if** $R \not\subseteq Q$ **then**
49 $Q \xleftarrow{+}$ IEMLReplace(R) ; // ajoute la solution R
50 return Q
51 **else**
52 **for** $\forall c \in C$ **do** // réduction de l'espace des solutions
53 **if** $\{\forall c_i \in C \backslash \{c\} \mid c \cap c_i\} = \emptyset$ **then**
54 $S \longleftarrow S \backslash c$; // enlève la multiplication
55 $C \longleftarrow C \backslash \{c\}$; // enlève la multiplication
56 $R \xleftarrow{+} c$; // ajoute la multiplication
57 **for** $\forall c \in C$ **do** // multiplications restantes
58 $\overline{S} \longleftarrow S \backslash c$; // duplique S, enlève c
59 $\overline{C} \longleftarrow C \backslash \{c\}$; // duplicate C, remove c
60 **for** $\forall c_i \in \overline{C}$ **do** // réduction de l'espace des solutions
61 **if** $c \cap c_i \neq \emptyset$ **then**
62 $\overline{C} \longleftarrow \overline{C} \backslash \{c_i\}$; // enlève les séquences invalides
63 $\overline{R} \longleftarrow R \cup \{c\}$; // duplique R, ajoute c
64 return SCRIPT_SOLVER$(\overline{C}, \overline{S}, \overline{R}, Q)$; // appel recursif

14 à 18) en ajoutant tous les membres qui n'ont pas de sèmes en commun à l'ensemble des solutions potentielles, et en les enlevant de l'ensemble des termes à considérer. Les lignes 19 à 25 établissent de nouvelles variables basées sur le reste de l'espace des solutions et font un appel récursif pour les traiter à la ligne 26.

La complexité spatio-temporelle de l'algorithme 4.4 est proportionnelle à :

$$l \cdot p \cdot \prod_{k=1}^{k=n/2} (n - 2k + 2) \tag{4.1}$$

où l est la longueur des séquences d'input, n est la taille de l'ensemble d'input C (cardinal de C), et p est la taille moyenne des éléments de C.

4.3 Du Script vers le Rhizome

Etant donné une expression Script en input, l'algorithme 4.5 crée un rhizome (un graphe dont les arêtes sont étiquetées). On notera que l'algorithme 4.5 appelle l'algorithme 4.6 qui, en retour, appelle l'algorithme 4.5. J'ai séparé la construction de rhizome en deux algorithmes distincts à seule fin d'augmenter la lisibilité du processus dans son ensemble. Les sections suivantes décrivent ce processus en détail.

La complexité des algorithmes 4.5 et 4.6 n'est pas analysée séparément puisqu'ils forment un seul processus. Ces deux algorithmes accomplissent ensemble la fonction suivante : l'expression Script en input est analysée afin de détecter les relations additives et multiplicatives ; au fur et à mesure du progrès de l'analyse, un arbre est construit, pourvu d'une certaine profondeur et d'une certaine largeur. La profondeur de l'arbre dépend du nombre de couches et la largeur de l'arbre dépend du nombre moyen de relations additives n_A sur chaque couche. De plus, la complexité de l'algorithme 4.6 dépends de la longueur moyenne de l'expression Script l_A du nombre moyen des expansions de caractère et de la complexité de l'algorithme 4.1. En tout et pour tout, la complexité est proportionnelle à $K \cdot n_A \cdot l_A \cdot e_A \cdot C_{SCRIPT\_GENERATOR}$, où K est un scalaire qui rend compte de toutes les vérifications d'appartenance à la liste et des fonctions globales, tandis que $C_{SCRIPT\_GENERATOR}$ représente la complexité de l'algorithme 4.1.

4.3.1 L'algorithme RHIZOME

L'algorithme 4.5 est utilisé sans aucun réglage spécial (voir la section 4.1.2) pour obtenir un rhizome. Le but de cet algorithme est de trouver toutes les relations additives et multiplicatives et d'identifier toutes les séquences à analyser par l'algorithme 4.6.

Les lignes 3 à 6 sont les conditions de terminaison pour l'algorithme. Le script d'input est alors analysé pour déterminer sa structure. Ou bien il n'y a pas de relations additives à la couche du script, ou bien il y a une – ou plus – relations additives. Dans le second cas, les relations additives sont ajoutées au rhizome (lignes 20 à 21) et l'on fait un appel récursif pour analyser à son tour chaque variable additive (lignes 22 à 23), ce qui déclenche la boucle logique à la ligne 8. Les lignes 8 à 18 accomplissent de nombreuses tâches, y compris l'analyse des bulbes et des relations multiplicatives. La ligne 9 déclenche l'algorithme 4.6 qui est décrit à la section 4.3.2. La structure du script est alors analysé pour détecter

les relations multiplicatives (ligne 10). S'il n'y a pas de relations multiplicatives, l'analyse prend fin et l'algorithme est terminé. Si une relation multiplicative a été trouvée, elle est ajoutée au rhizome (lignes 12 à 16). On fait un appel récursif pour analyser chaque variable multiplicative à son tour (lignes 17 à 18).

Algorithm 4.5 Algorithme RHIZOME

input : (s) où s est une expression Script
output : variable globale mise à jour $G = (V, E)$

```
65 A, B ;                                   // ensemble d'expressions Script
66 begin
67     if Layer(s) < MinLayer then
68         return ;                                          // fin de l'analyse
69     if s ∈ Terms ∪ RestrictedScript then
70         return ;                                          // fin de l'analyse
71     A ⟵ Parse(s, ⊕) ;                                  // relations additives
72     if |A| = 0 then
73         SCRIPT_DECOMPRESSOR(s) ;                          // analyse de bulbe
74         B ⟵ Parse(s, ⊗) ;                          // relations multiplicatives
75         if |B| > 0 then
76             if B ∩ RestrictedScript = ∅ then
77                 G(V, E) ⟵+ MultiplicationOrder(s, B)
78             if MultSym then
79                 if ∃ v ∈ G = (V, E) | v = RoleInterversion(s) then
80                     G(V, E) ⟵+ MultiplicationSymmetry(s, {v})
81             for b ∈ B do
82                 RHIZOME(b) ;                  // analyse des couches inférieures
83     else
84         if A ∩ RestrictedScript = ∅ then
85             G(V, E) ⟵+ AdditionRelations(s, A)
86         for a ∈ A do
87             RHIZOME(a)
```

4.3.2 L'algorithme SCRIPT_DECOMPRESSOR

Basé sur la décomposition de caractères, l'algorithme 4.6 crée de nouvelles expressions Script à partir de l'expression Script d'input et ajoute des relations additives entre les nouvelles expression Script. Chaque nouvelle expression Script devient un input de l'algorithme 4.5 qui va l'analyser.

Le script d'input est analysé pour déterminer tous les caractères de couche 0, qui sont enregistrés dans la structure de données `Chars` (ligne 5). Chaque

Algorithm 4.6 Algorithme SCRIPT_DECOMPRESSOR

input : (s) où s est une expression Script
output : variable globale mise à jour $G = (V, E)$

```
88 A ;                                  // structure de données de type Chars
89 B ;                                       // ensemble d'expressions Script
90 C ;                                              // ensemble de séquences
91 begin
92     A ⟵ Parse(s, ©)
       for 0 ≤ i < |A| do
93         B ⟵ ∅
           C ⟵ Expand(A[i])
           for 0 ≤ j < |C| do
94             Ā ⟵ A
               Ā[i] ⟵ C[j]
               B ⟵+ SCRIPT_GENERATOR(CreateAlgebraic(Ā))
95         if |B| > 0 then
96             if B ∩ RestrictedScript = ∅ then
97                 G(V, E) ⟵+ AdditionRelations(s, B)
98             for b ∈ B do
99                 RHIZOME(b)
```

caractère est alors analysé à son tour (lignes 6 à 17). Chaque caractère est décompressé (ligne 8) et les caractères décompressés remplacent les caractères à partir desquels ils ont été obtenus (lignes 10 à 11). Une expression Script valide est régénérée à partir de l'expression Script modifiée (qui peut être invalide) et enregistrée dans une variable temporaire B (ligne 12). Si B ne contient aucun membre, l'analyse s'arête et l'algorithme se termine. Sinon, des relations additives entre les Scripts modifiés sont ajoutés au rhizome (ligne 15) et on fait appel à l'algorithme 4.5 (ligne 17).

4.3.3 Construction de clés

Les algorithmes 4.5 et 4.6 sont utilisés avec les réglages suivants pour obtenir des clés :

1. Le réglage du paramètre `MinLayer` à $L_n - 1$ va limiter l'analyse au calcul est des relations d'ordre et de symétrie additive pour la couche L_n et les relations d'ordre multiplicative pour la couche $L_n - 1$. Les lignes 3 à 4 de l'algorithme 4.5 dépend de ce réglage.
2. Le réglage du paramètre `MultSym` à logiquement $faux$, va désactiver la création des relations de symétrie multiplicative. Cette caractéristique est contrôlée par les lignes 14 à 16 de l'algorithme 4.5 dépends de ce réglage.

3. Le réglage du paramètre `MultOrd` à l'une des valeurs permises va contrôler les relations d'ordre multiplicatif qui seront créées (voir la section 4.1.3, en particulier la fonction `MultiplicationOrder`).
4. Le paramètre `RestrictedScript` définit une liste de scripts restreints. Les scripts qui appartiennent à cette liste ne doivent pas être analysés. Les lignes 5, 12, 20 de l'algorithme 4.5 et la ligne 14 et de l'algorithme 4.6 contrôlent cette caractéristique.

4.4 Du Rhizome vers la Sphère sémantique

La typologie des mots, des clauses et des phrases, ainsi que les constructions non-grammaticales ont été expliquées plus haut en 3.2.8. Etant donnée une expression du Script d'IEML s, l'algorithme 4.7 est utilisé pour générer toutes les clauses et toutes les phrases. L'algorithme 4.7 délègue les calculs aux algorithmes 4.5, 4.8, 4.9 et 4.10. La complexité de l'algorithme 4.7 est donc proportionnelle à la complexité des algorithmes sous-jacents. L'algorithme 4.7 utilise l'algorithme 4.5 avec le réglage suivant :

— Le paramètre `Terms` définit une liste d'expressions Script. Chaque script qui appartient à cette liste *ne doit pas* être analysé. Les lignes 5 à 6 de l'algorithme 4.5 dépendent de ce réglage du paramètre `Terms`.

Algorithm 4.7 Algorithme ENUNCIATION

input : (s) où s est une expression Script
output : Variables globales $G = (V, E)$, W', W'', C^1, C^{k+1} et P^k

```
100 begin
101     G = RHIZOME(s) ;                          // calcule G = (V, E)
102     WORDS(G) ;                                // calcule W' et W''
103     CLAUSE(G) ;                               // calcule C^1
104     PHRASE(G) ;                               // calcule C^{k+1} et P^k
```

Les sections qui suivent décrivent les algorithmes 4.8, 4.9 et 4.10. Pour une description de l'algorithme 4.5, on se réfèrera à la section 4.3.

4.4.1 L'algorithme WORDS

L'algorithme 4.8 traverse le rhizome afin de détecter des mots (*words* en anglais). Il y a deux types de mots, enregistrés dans $\mathtt{W}'$ et $\mathtt{W}''$, qui diffèrent par le type de relations auxquels ils participent et par leur pattern d'expression. La traversée de rhizome elle-même est implémenté dans l'algorithme 4.11 (lignes 2 et 4). Les résultats de la traversée sont alors transformés par une fonction globale aux lignes 3 et 5. La complexité de cet algorithme est directement proportionnelle à la complexité de l'algorithme 4.11.

Algorithm 4.8 Algorithme WORDS

input : (G) où G est un graphe $G = (V, E^A \cup E^B \cup E^C \cup E^D)$
output : Variables globales W' et W''

```
105 begin
106   TRAVERSE(G, V, E^A, E^B, W', Terms) ;          // traverse le graphe
107   W' ⟵ {w ∈ W' | DetType(w)} ;                   // transforme le résultat
108   TRAVERSE(G, Pattern('SES'), E^C, E^D, W'', Terms)
      W'' ⟵ {w ∈ W'' | DetType(w)} ;                 // transforme le résultat
```

$\texttt{C}^1$
$SEE \otimes EEE \otimes SES$
$SES \otimes EEE \otimes SEE$
$SES \otimes EEE \otimes SES$
$SES \otimes EEE \otimes EEE$
$S \otimes S \otimes SES$
$S \otimes SES \otimes S$
$S \otimes SES \otimes SES$
$SES \otimes S \otimes S$
$SES \otimes S \otimes SES$
$SES \otimes SES \otimes S$
$SES \otimes SES \otimes SES$

TABLE 4.3 – Modèle (*template*) de la fonction *Pattern* pour le paramètre $\texttt{C}^1$

4.4.2 L'algorithme CLAUSE

Le but de l'algorithme 4.9 est de trouver toutes les clauses de couche 1. La chose est accomplie en considérant différents points de départ pour la traversée du rhizome (ou bien des membres de `Terms` ou bien des membres de $\texttt{W}' \cup W''$), et en cherchant les scripts qui se conforment à un pattern particulier (en fournissant différents modèles pour la fonction `Pattern`) aux lignes 2 et 3. La complexité de cet algorithme est directement proportionnelle à la complexité de l'algorithme 4.11.

Algorithm 4.9 Algorithme CLAUSE

input : (G) où G est un graphe $G = (V, E^A \cup E^B \cup E^C \cup E^D)$
output : Variable globale C^1

```
109 begin
110   TRAVERSE(G, Pattern('SES' | 'SEE'), E^C, E^D, C^1, Terms)
      TRAVERSE(G, Pattern('clause'), E^C, E^D, C^1, W' ∪ W'')
```

4.4.3 L'algorithme PHRASE

Le but de l'algorithme 4.10 est de trouver toutes les clauses de couche 2 et au-dessus, ainsi que toutes les phrases de couche 1 et au-dessus.

L'algorithme 4.10 calcule toutes les phrases (ligne 3), tandis que la ligne 5 calcule toutes les clauses de couche 2 et au-dessus. La logique de cet algorithme est très semblable à celle de l'algorithme 4.11. Sa complexité spatio-temporelle est un multiple scalaire (proportionnel au nombre de couches) de la complexité de l'algorithme 4.11.

Algorithm 4.10 Algorithme PHRASE

input : (G) où G est un graphe $G = (V, E^A \cup E^B \cup E^C \cup E^D)$
output : Variables globales C^{k+1} et P^k

```
111 begin
112     for 1 ≤ i ≤ 6 do
113         TRAVERSE(G, V, E^A, E^B, P^i, C^i)
            if i < 6 then
114             TRAVERSE(G, Pattern(‘SSS‘ | ‘SES‘ | ‘SEE‘), E^C, E^D, C^{i+1}, P^i)
```

4.4.4 L'algorithme TRAVERSE

L'algorithme 4.11 est un algorithme auxiliaire générique, dont le rôle est de traverser un graphe et de grouper les sommets qui se conforment à certains critères. Cette opération revient plusieurs fois dans le processus de transcodage et c'est pourquoi il est utile de l'abstraire afin de rendre les algorithmes 4.8 , 4.9 et 4.10 plus lisibles.

Algorithm 4.11 Algorithme TRAVERSE

input : $(G, A_p, A_q, A_r, A_s, A_w)$ où G est un graphe $G = (V, E)$ et A_p, A_q, A_r, A_s, A_w sont des ensembles
output : variable mise à jour pointée par A_s

```
115 begin
116     for ∀w ∈ A_w do
117         V_r ⟵ {v ∈ V | {w, v} ∈ A_r} ;                  // sous-ensemble de V
118         V^w_{A_q} ⟵ {v ∈ V | v ∉ A_s ∧ (v, w) ∈ A_q} ;  // sous-ensemble de V
119         for ∀v^w_{A_q} ∈ V^w_{A_q} do
120             V_q ⟵ {v ∈ V | (v^w_{A_q}, v) ∈ A_q} ;      // sous-ensemble de V
121             if V_q ∩ V_r ⊆ A_w then
122                 if v^w_{A_q} ∈ A_p then
123                     A_s ⟵^+ v^w_{A_q}
```

L'algorithme 4.11 fonctionne pour tous les membres w de l'ensemble A_w (ligne 2). Les variables locales V_r et $V_{A_q}^w$ sont initialisées respectivement avec tous les sommets du graphe dont les relations avec w sont contenues dans A_r, et avec tous les sommets du graphe qui ne sont pas présents dans A_s et dont la relation avec w est contenue dans A_q (lignes 3 et 4). L'algorithme commence son itération sur tous les membres de $V_{A_q}^w$ (ligne 5). Une variable locale V_q enregistre tous les sommets du graphe dont les relations avec $v_{A_q}^w$ sont contenues dans A_q (ligne 6). A l'étape finale, l'algorithme enregistre une variable $v_{A_q}^w$ dans l'ensemble A_s si l'ensemble $V_q \cap V_r$ est un sous-ensemble de l'ensemble A_w et si la variable $v_{A_q}^w$ appartient à l'ensemble A_p (lignes 7 à 9).

Puisqu'il s'agit d'un algorithme générique, les contraintes suivantes doivent être respectées :

1. Les membres de V_q, V_r, A_w, A_p et A_s sont du même type que $v_{A_q}^w$
2. Les membres deA_r sont des ensembles
3. Les membres de A_q sont des paires ordonnées

Nous sommes ici dans le contexte du processus de transcodage. Cela signifie que les membres des variables dans la première contrainte sont des sommets. Les membres de la variable de la seconde contrainte sont des relations de symétrie additive ou multiplicative (arêtes du graphe). Finalement les membres de la variable de la troisième contrainte sont des relations d'ordre multiplicatif ou additif, qui sont représentées par des arêtes orientées puisqu'elles dénotent des relations parent-enfant.

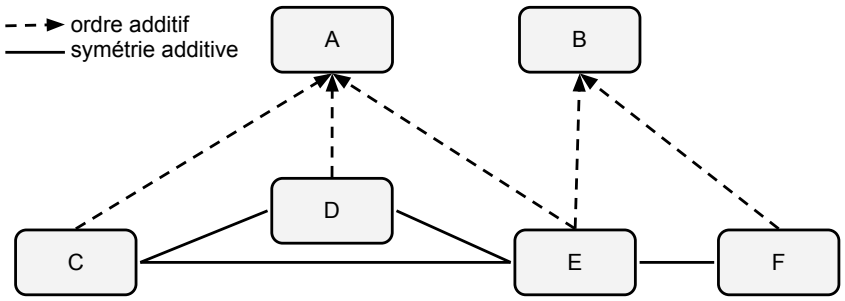

FIGURE 4.1 – Relations d'ordre et de symétrie additives

Considérons par exemple le sommet E dans la figure 4.1. Etant donné un graphe

$$G = (A_w, A_r \cup A_q) \tag{4.2}$$

où

$$A_w = \{A, B, C, D, E, F\} \tag{4.3}$$

$$A_r = \{\{C, D\}, \{C, E\}, \{D, E\}, \{E, F\}\} \tag{4.4}$$

$$A_q = \{(A, C), (A, D), (A, E), (B, E), (B, F)\} \tag{4.5}$$

and $A_s = \emptyset$, $A_p = A_w$.

L'exécution de l'algorithme 4.11 donne le résultat suivant : $V_r = C, D, F$ (ligne 3), $V^w_{A_q} = A, B$ (ligne 4). Nous avons maintenant 2 cas à considérer : dans le premier cas, $v^w_{A_q} = A$, $V_q = C, D, E$, les conditions sur les lignes 7 et 8 sont respectées, et nous enregistrons A dans A_s; dans le second cas, $v^w_{A_q} = B$, $V_q = E, F$, les conditions sur les lignes 7 et 8 sont respectées, et nous enregistrons B dans A_s. Si cependant F n'était pas contenu dans A_w, ce serait A et non B qui serait enregistré dans A_s.

Supposons que les ensembles soient implémentés comme des tables de hachage, la complexité temporelle de l'algorithme 4.11 est :

$$T_1 = |V^w_{A_q}| * ((min(|A_q|, |V|) + min(|V_q|, |V_r|, |A_w|)) \tag{4.6}$$

pour les lignes 5 à 9, et :

$$T_2 = |A_w| * (min(|A_r|, |V|) + min(|A_q|, |V|) + T_1) \tag{4.7}$$

pour la totalité. Notons que $|...|$ représente la taille d'un ensemble particulier. Une limite supérieure pour cette équation est donnée par $|V|^3$ lorsque V devient grand. La complexité de l'espace est alors proportionnelle à $|V|$.

4.4.5 L'algorithme SEMANTIC_NETWORK

L'algorithme 4.12 crée les structures de données nécessaires à la représentation du réseau sémantique sous forme visuelle. Ces structures de données contiennent les éléments suivant :

- une représentation de chaque terme dans une langue naturelle sélectionnée par l'intermédiaire d'un dictionnaire spécifié (ligne 3),

- trois graphes qui définissent respectivement les relations entre termes (ligne 5), les relations entre graphes (ligne 9 et 14) et finalement un graphe qui peut être disjoint (ligne 17).

Ces structures de données définissent l'information nécessaire et suffisante pour représenter un réseau sémantique.

La ligne 5 ajoute les sommets et les arêtes représentant un graphe complet entre toutes les séquences en relation de symétrie additive. La ligne 9 ajoute les sommets et les arêtes, la ligne 14 ajoute les sommets et les arêtes étiquetées, les sommets étant les sèmes substance et attributs d'une séquence et l'étiquette de l'arête est le mode de la séquence (si le sème en rôle de mode est vide, il peut alors être omis). Finalement, la ligne 17 regroupe des graphes spécifiques.

Algorithm 4.12 Algorithme SEMANTIC_NETWORK

input : (T) où T est le "mapping" en langue naturelle
output : variables globales $G_{SEM}(V,E)$, $\bar{G}_{SEM}(V,E)$, $\bar{\bar{G}}_{SEM}(V,E)$

```
124 begin
125     for ∀t ∈ Terms do                                   // tous les terms
126         H_NAT ←+ {t, Translate(t, T)} ;   // mapping des termes vers le
            langage
127     for ∀w ∈ W' do
128         G_SEM ←+ Clique(w) ;                  // Graphe complet de w
129     for ∀w ∈ W'' do
130         v ⟵ {w_sub, w_mod} ;                                  // sommet
131         e ⟵ {(w_sub, w_mod)} ;                                // arête
132         Ḡ_SEM(V,E) ←+ (v, e)
133     for ∀c ∈ ⋃_{i=0}^{6} C^i do
134         v ⟵ {w_sub, w_att} ;                                  // sommet
135         e ⟵ {(w_sub, w_att)} ;                                // arête
136         e ←+ (w_mod) ;                          // étiquette d'arête
137         Ḡ_SEM(V,E) ←+ (v, e)
138     for ∀p ∈ {P^i | 1 ≤ i ≤ 6} do
139         for ∀c ∈ p do
140             B̄̄_SEM(p) ←+ Ḡ(c)
```

Chapitre 5

Formalisation mathématique

« Nul n'entre ici s'il n'est géomètre » Platon

5.1 Introduction

Ce chapitre présente les aspects mathématiques du métalangage de l'économie de l'information.

Le modèle formel du langage IEML et de ses variables sémantiques est exposé dans la section 5.2. On montre qu'IEML est un langage régulier, une classe de langages qui est extrêmement efficace pour les calculs impliquant la répétition et le séquençage et qui est en outre reconnue par des *machines à états finis*.

La notion de *structure de groupe* est liée au concept mathématique de symétrie, qui peut être compris comme une invariance par transformation. L'étude des symétries permet la reconnaissance de similarités et le repérage des propriétés des éléments qui ne changent pas malgré leurs transformations. On montre dans la section 5.3 que les structures de *groupes* et d'*anneaux* sont inhérentes au langage IEML et que le langage IEML peut lui-même être considéré comme une catégorie, c'est-à-dire un méta-groupe de transformations dans le vocabulaire technique des mathématiques.

Dans la section 5.4 on considère la *calculabilité* des transformations appliquées aux expressions du langage IEML. En utilisant les machines à état fini comme modèle de calcul sous-jacent, on montre que les transformations définies sur les expressions du langage IEML sont calculables.

Alors que le *langage* IEML est utilisé pour créer des *expressions*, les *graphes* sémantiques représentent les *relations sémantiques* qui entretissent les expressions d'IEML. Le modèle des relations entre expressions IEML est présenté dans la section 5.5.

Les *circuits sémantiques* résultent de la combinaison de plusieurs graphes sémantiques. Dans la section 5.6, on montre que les circuits sémantiques forment un groupoïde, et des algorithmes pour des fonctions applicables à ces circuits sont présentées.

Finalement, ce chapitre se termine sur la section 5.7 qui contient la description de méthodes, empruntées à la théorie spectrale des graphes, pour calculer des distances sémantiques dans et entre les circuits sémantiques.

Je voudrais remercier Andrew Roczniak, PhD, qui m'a aidé pendant près de dix années à formaliser IEML et ses circuits ainsi que Nick Soveiko, PhD, qui a relu la version finale de ce chapitre mathématique et a proposé l'utilisation des espaces propres de graphes pour calculer des similarités entre circuits sémantiques. L'auteur est néanmoins le seul responsable de toute erreur ou incohérence que le lecteur attentif pourrait découvrir.

5.2 Le langage régulier IEML

5.2.1 Modèle du langage

Σ est un ensemble de symboles fini et non-vide tel que $\Sigma = \{S, B, T, U, A, E\}$.

Considérons une chaîne s qui soit une séquence finie de symboles de $\bar{\Sigma}$. La longueur de cette chaîne est dénotée par $|s|$. Une chaîne vide ϵ est une chaîne avec zéro occurrences de symboles et sa longueur est $|\epsilon| = 0$. L'ensemble de toutes les chaînes de longueur k qui sont composées avec des symboles de Σ est définie comme $\Sigma^k \triangleq \{s \text{ où } |s| = k\}$. L'ensemble de toutes les chaînes tirées de Σ est défini comme :

$$\Sigma^* \triangleq \Sigma^0 \cup \Sigma^1 \dots \tag{5.1}$$

Le langage IEML sur Σ est un sous-ensemble de Σ^*, $L_{IEML} \subseteq \Sigma^*$ où $L = 6$:

$$L_{IEML} \triangleq \left\{ s \in \Sigma^* \mid 0 \leq l \leq L,\ |s| = 3^l \right\} \tag{5.2}$$

Proposition 5.2.1. *Le langage IEML donné dans l'équation 5.2 est un langage régulier [KLE 1956].*

Démonstration. Considérons la définition des langages réguliers :

- $L = \{\emptyset\}$ et $L = \{\epsilon\}$ sont des langages réguliers,
- $L = \{\sigma \mid \sigma \in \Sigma\}$ sont des langages réguliers,
- si L_1 et L_2 sont des langages réguliers, alors $L_1 \cup L_2$ et $L_1 \cdot L_2$ (concaténation) le sont aussi.

Puisque L_{IEML} peut être construit à partir de son alphabet $\Sigma = \{S, B, T, U, A, E\}$ *et* en utilisant seulement des propositions issues de la définition qui vient d'être donnée, c'est un langage régulier. □

Notons que chaque chaîne appartenant au langage L_{IEML} peut aussi être obtenue à partir de l'alphabet Σ par l'application de la fonction de *triplication*. L'opération de concaténation prend deux chaînes et en produit une troisième qui est composée des symboles de la première chaîne suivis des symboles de la seconde chaîne. La fonction de triplication en IEML est une spécialisation de l'opération de concaténation, dans laquelle trois chaînes (a, b, c) de même

longueur appartenant à L_{IEML} sont concaténées et où la longueur de chaque chaîne est au maximum 3^{L-1}:

$$f_t(a,b,c) \triangleq (abc \mid |a| \leq 3^{L-1} \wedge\ |a| = |b| = |c|\ \wedge\ a,b,c \in L_{IEML}) \tag{5.3}$$

Comme on l'a dit en introduction de ce chapitre, les langages réguliers représentent une classe de langages qui est extrêmement efficace pour les calculs impliquant la répétition et le séquençage. Les langages réguliers sont en outre reconnus par les *machines à états finies*, qui sont discutées à la section5.4.

5.2.2 Modèle des séquences sémantiques

Definition 5.2.1. *Une chaîne s est une séquence sémantique si et seulement si* $s \in L_{IEML}$.

Sauf mention spéciale, 'séquence' et 'séquence sémantique' sont utilisées de manière interchangeable dans la suite de ce texte mathématique. Pour dénoter le $p_n ième$ symbole d'une séquence s, j'utilise l'exposant n pour lequel $1 \leq n \leq 3^l$ et j'écris s^n. Notons que pour chaque séquence s de couche l, s^n n'est pas défini pour tout $n > 3^l$. [1] Deux séquences sémantiques sont distinctes si et seulement si les conditions suivantes sont remplies: a) leurs couches sont différentes, b) elles sont composées de caractères différents, c) leurs caractères ne suivent pas le même ordre : quel que soit s_a et s_b,

$$s_a = s_b \iff \forall n,\ s_a^n = s_b^n\ \wedge\ |s_a| = |s_b| \tag{5.4}$$

Considérons maintenant les relations binaires entre séquences sémantiques en général. Elles sont obtenues en effectuant un produit cartésien de deux ensembles. [2] Pour chaque ensemble de séquences sémantiques X, Y où $s_a \in X$, $s_b \in Y$ et en utilisant l'équation 5.4, nous définissons quatre relations binaires *tout* $\subseteq X \times Y$, *substance* $\subseteq X \times Y$, *attribut* $\subseteq X \times Y$ et *mode* $\subseteq X \times Y$ comme suit :

$$\texttt{tout} \triangleq \{(s_a, s_b) \mid s_a = s_b\} \tag{5.5}$$

$$\texttt{substance} \triangleq \{(s_a, s_b) \mid 1 \leq n \leq |s_b|,\ s_a^n = s_b^n\ \wedge\ |s_a| = 3|s_b|\} \tag{5.6}$$

$$\texttt{attribut} \triangleq \{(s_a, s_b) \mid 1 \leq n \leq |s_b|,\ s_a^{n+|s_b|} = s_b^n\ \wedge\ |s_a| = 3|s_b|\} \tag{5.7}$$

$$\texttt{mode} \triangleq \{(s_a, s_b) \mid 1 \leq n \leq |s_b|,\ s_a^{n+2|s_b|} = s_b^n\ \wedge\ |s_a| = 3|s_b|\} \tag{5.8}$$

1. Sur la notion de couche, voir la sous-section 5.2.3.
2. Un produit cartésien de deux ensembles X et Y s'écrit de la manière suivante: $X \times Y = \{(x,y) \mid x \in X, y \in Y\}$

L'équation 5.5 établit que deux séquences sémantiques égales sont en relation de `tout` (on peut aussi écrire s_b `tout` s_a). Les équations 5.6, 5.7 et 5.8 établissent que deux séquences sémantiques qui partagent des sous-séquences spécifiques peuvent être en relation de `substance`, `attribut` ou `mode`. Quelles que soient s_a et s_b, si elles sont dans une des relations définies plus haut, alors nous disons que s_b *joue un rôle eu égard à* s_a et nous appelons s_b un *sème* de la séquence s_a :

Definition 5.2.2. *Quelles que soient les séquences sémantiques s_a et s_b, si $(s_a, s_b) \in$ `whole` $\cup$ `substance` $\cup$ `attribute` $\cup$ `mode`, alors s_b joue un rôle* dans *s_a et s_b est appelé un sème.*

Nous pouvons maintenant regrouper les séquences sémantiques en ensembles. Un regroupement particulièrement utile est basé sur la *couche* (ou la longueur) de ces séquences, tel que discuté dans la sous-section 5.2.3 qui suit.

5.2.3 Modèle des catégories sémantiques

Une *catégorie* de L_{IEML} de couche l, est un sous-ensemble de L_{IEML} tel que toutes les séquences sémantiques de ce sous-ensemble ont la même longueur :

$$c_l \triangleq \left\{ s \mid s \in L_{IEML} \ \wedge \ |s| = 3^l \right\} \tag{5.9}$$

Definition 5.2.3. *Une* catégorie sémantique *c est un ensemble contenant des séquences sémantiques de même couche.*

Sauf précision particulière 'catégorie' et 'catégorie sémantique' sont utilisées de manière interchangeable dans la suite de ce texte. La couche de chaque catégorie c est exactement la même que la couche des séquences sémantiques inclues dans cette catégorie. L'ensemble de toutes les catégories de couche L est définie comme l'ensemble des parties [3] de l'ensemble de toutes les séquences de couche L de L_{IEML}:

$$C_L \triangleq \mathcal{P}(\{c_L\}) \tag{5.10}$$

Deux catégories sont distinctes si et seulement si elles diffèrent d'au moins un élément. Quelles que soient c_a et c_b:

$$c_a = c_b \iff c_a \subseteq c_b \wedge c_b \subseteq c_a \tag{5.11}$$

Une condition plus faible peut être appliquées aux catégories de couches distinctes (puisque deux catégories sont différentes si leurs couches sont différentes) et s'écrit ainsi :

$$\ell(c_a) \neq \ell(c_b) \Longrightarrow c_a \neq c_b \tag{5.12}$$

où $\ell(\cdot)$ dénote la couche d'une catégorie.

3. L'ensemble des parties de S est l'ensemble de tous les sous-ensembles de S, y compris l'ensemble vide $\emptyset$

De manière analogue aux séquences, considérons des relations binaires entre des *catégories* c_i et c_j pour qui $\ell(c_i), \ell(c_j) \geq 1$. Quelles que soient les catégories X, Y dans lesquelles $c_a \in X$, $c_b \in Y$ et en utilisant les équations 5.11, 5.6, 5.7 et 5.8 nous définissons quatre relations binaires $tout_C \subseteq X \times Y$, $substance_C \subseteq X \times Y$, $attribut_C \subseteq X \times Y$ et $mode_C \subseteq X \times Y$ comme suit :

$$\texttt{tout}_C \triangleq \{(c_a, c_b) \mid c_a = c_b\} \tag{5.13}$$

$$\texttt{substance}_C \triangleq \{(c_a, c_b) \mid \forall\ s_a \in c_a, \exists s_b \in c_b, (s_a, s_b) \in \texttt{substance}\} \tag{5.14}$$

$$\texttt{attribut}_C \triangleq \{(c_a, c_b) \mid \forall\ s_a \in c_a, \exists s_b \in c_b, (s_a, s_b) \in \texttt{attribut}\} \tag{5.15}$$

$$\texttt{mode}_C \triangleq \{(c_a, c_b) \mid \forall\ s_a \in c_a, \exists s_b \in c_b, (s_a, s_b) \in \texttt{mode}\} \tag{5.16}$$

Quelles que soient les deux catégories c_a, c_b, si elles sont ensemble dans une des relations ci-dessus, alors nous disons que c_b joue un *rôle* par rapport à c_a et c_b est appelé un sème de la catégorie c_a :

Definition 5.2.4. *Quelles que soient les* catégories c_a *et* c_b*, si* $(c_a, c_b) \in \texttt{tout}_C \cup \texttt{substance}_C \cup \texttt{attribut}_C \cup \texttt{mode}_C$*, alors* c_b *joue* un *rôle eu égard à* c_a *et nous appelons* c_b *un* sème *de* c_a*.*

5.2.4 Langage des catégories

Les catégories seront habituellement générées selon l'approche suivante. Prenons d'abord l'ensemble des parties de Σ (l'ensemble de tous les sous-ensembles de Σ, incluant l'ensemble vide $\emptyset$). Nous le représentons par : $\Sigma_{IEML} \triangleq \mathcal{P}(\Sigma)$. Σ_{IEML} contient des ensembles tels que $\{S\}$, $\{A\}$, $\{U, A\}$, $\{E\}$, $\{S, B, T\}$, $\{\emptyset\}$. Notons que l'ordre à l'intérieur des membres de l'ensemble Σ_{IEML} n'est pas pertinent. Par exemple les symboles $\{U, A\}$ et $\{A, U\}$ sont considérés comme identiques. Σ_{IEML} est l'ensemble de tous les caractères sémantiques IEML. Nous définissons le langage des catégories comme le langage utilisant l'alphabet suivant :

$$\bar{\Sigma} \triangleq \Sigma_{IEML} \backslash \{\emptyset\} \tag{5.17}$$

Soit s une chaîne finie de symboles choisis dans $\bar{\Sigma}$. La longueur de cette chaîne est notée par $|s|$. Une chaîne vide ϵ est une chaîne avec zéro occurrence de symboles et sa longueur est : $|\epsilon| = 0$. L'ensemble de toutes les chaînes de longueur k composées de symboles de $\bar{\Sigma}$ est défini comme $\bar{\Sigma}^k \triangleq \{s : |s| = k\}$. L'ensemble de toutes les chaînes prises sur $\bar{\Sigma}$ se définit ainsi :

$$\Sigma^*_{cat} \triangleq \bar{\Sigma}^0 \cup \bar{\Sigma}^1 \ldots \tag{5.18}$$

Le langage des catégories pris sur $\bar{\Sigma}$ est un sous-ensemble Σ^*_{cat}, $L_{cat} \subseteq \Sigma^*_{cat}$ où $L = 6$:

$$L_{cat} \triangleq \left\{ s \in \Sigma^*_{cat} \mid 0 \leq l \leq L,\ |s| = 3^l \right\} \tag{5.19}$$

Les séquences d'IEML sont obtenues à partir de n'importe quelle séquence de $s \in L_{cat}$ en effectuant un produit cartésien entre les symboles de la séquence s du langage des catégories. Par exemple, la séquence $\{S, B, T\}\{U, A\}\{E\}$ du langage des catégories donne l'ensemble des séquences IEML qui suit : $\{SUE, SAE, BUE, BAE, TUE, TAE\}$.

5.2.5 Types de catégories sémantiques

Introduisons maintenant certains *types* de catégories :

Definition 5.2.5. *Une* catégorie c *de couche* l *est* singulière *aux conditions suivantes :* $|c| = 1$ *et la séquence* $s \in c$ *est composée de symboles* s^n *où* $s^n \in \bar{\Sigma} \wedge |s^n| = 1$ *pour* $1 \leq n \leq 3^l$.

Toutes les *catégories* qui ne sont pas singulières sont *plurielles*.

Definition 5.2.6. *Une catégorie* c *de couche* l *est* simple *aux conditions suivantes :* $|c| = 1$ *et la séquence* $s \in c$ *est composée de symboles* s^n *où* $s^n \in \bar{\Sigma}$ *pour* $1 \leq n \leq 3^l$.

Toutes les catégories qui ne sont pas simples sont *complexes*. Notons que toutes les catégories singulières sont aussi simples, puisque $\bar{\Sigma}$ contient tous les symboles qui ne contiennent qu'un seul membre ($\{S\}$, $\{B\}$, $\{T\}$, $\{U\}$, $\{A\}$, $\{E\}$).

5.2.6 Modèle des Catsets et des USL

Un catset est un ensemble de catégories distinctes de même couche :

Definition 5.2.7. *Un catset* κ *est un ensemble contenant des catégories* $\kappa_l = \{c \mid l(c) = l\}$ *tel que* $\forall a, b \in \kappa_l,\ a \neq b$

La couche d'un catset [4] est donnée par la couche de n'importe lequel de ses membres : si une $c \in \kappa$, alors $\ell(\kappa) = \ell(c)$. Un USL est composé d'un ensemble d'au maximum sept catsets de couches différentes :

Definition 5.2.8. *Un USL* u *est un ensemble contenant des catsets de différentes couches :* $u = \{\kappa\}$ *tel que* $\forall a, b \in u,\ l(a) \neq l(b)$

Il faut noter que puisqu'il a 7 couches distinctes, un USL peut avoir au plus sept membres. Toutes les opérations ensemblistes standard sur les USL sont toujours exécutées sur des ensembles de *catégories* (et donc sur des ensembles de séquences), couche par couche.

4. Notez qu'une *catégorie* c peut être écrite comme $c \in C_L$, alors qu'un catset κ peut être écrit comme $\kappa \subseteq C_L$

5.3 Les propriétés de symétrie d'IEML

5.3.1 Généralités

Le concept mathématique de *groupe* se définit par un *ensemble,* une *opération binaire* et des *propriétés* de l'opération appliquée aux membres de l'ensemble. L'opération binaire $\circledast$ sur un ensemble S associe aux éléments x et y de S un troisième élément $x \circledast y$ de S. Les propriétés sont l'associativité ($\forall x, y, z \in S,\ (x \circledast y) \circledast z = x \circledast (y \circledast z)$), l'existence d'un élément neutre et l'existence d'un élément symétrique pour chacun des éléments de l'ensemble.

Notons les éléments neutre et inverses d'un groupe S par x^1 et x^{-1}. Les propriétés fondamentales des groupes sont les suivantes.

Proposition 5.3.1. *Un groupe $(S, \circledast)$ a exactement un élément neutre x^1 satisfaisant à la condition suivante : $\forall x \in S,\ x \circledast x^1 = x^1 \circledast x = x$*

Démonstration. Supposons qu'un élément $i \in S$ ait la propriété $i \circledast x = x,\ \forall x \in S$. Nous avons alors $i = i \circledast x^1 = x^1$. De façon similaire, si $i \in S$ a la propriété $x \circledast i = x,\ \forall x \in S$, alors nous obtenons $i = x^1 \circledast i = x^1$. □

Proposition 5.3.2. *Un groupe $(S, \circledast)$ a exactement un élément symétrique x^{-1} pour tout $x \in S$*

Démonstration. Il découle des propriétés du groupe qu'il existe un élément $x^{-1} \in S$ avec la propriété $x \circledast x^{-1} = x^{-1} \circledast x = x^1,\ \forall x \in S$. Si $i \in S$ a la propriété $x \circledast i = x^1$, alors $i = x^1 \circledast i = (x^{-1} \circledast x) \circledast i = x^{-1} \circledast (x \circledast i) = x^{-1} \circledast x^1 = x^{-1}$. De manière similaire, pour chaque élément ayant la propriété $i \circledast x = x^1$, alors $i = x^{-1}$, ce qui implique que le symétrique de chaque $x \in S$ est déterminé de façon unique. □

Proposition 5.3.3. *Quels que soient les éléments x, y d'un groupe $(S, \circledast)$, $(x \circledast y)^{-1} = y^{-1} \circledast x^{-1}$*

Démonstration. Nous vérifions que $(x \circledast y) \circledast (y^{-1} \circledast x^{-1}) = x \circledast (y \circledast (y^{-1} \circledast x^{-1})) = x \circledast (y \circledast y^{-1} (\circledast x^{-1})) = x \circledast (x^1 \circledast x^{-1}) = x \circledast x^{-1} = x^1$. Un résultat similaire est obtenu pour $(y^{-1} \circledast x^{-1}) \circledast (x \circledast y)$, ce qui implique que le symétrique de $(x \circledast y)$ est $y^{-1} \circledast x^{-1}$. □

La structure de groupe peut être étendue en ajoutant des opérations, ce qui mène à une structure à deux opérations appelée un anneau ; ou en ajoutant des propriétés (par exemple, l'ajout de la commutativité - $x \circledast y = y \circledast x,\ \forall x, y \in S$ - crée un groupe *abélien*). Enlever ou limiter certaines propriétés mène à des structures plus "simples" appelées *groupoïdes.*

5.3.2 Catégories sémantiques

Pour n'importe quelle catégorie c nous pouvons définir une fonction f de c dans c, $f : c \to c$ qui est injective, $\forall x, y \in c, f(x) = f(y) \to x = y$ et totale,

$\forall x \in c,\ f(x) \in c$. Nous rassemblons toutes ces fonctions distinctes dans un ensemble :

$$F_c = \{f_n \mid \exists s \in c, i \neq j, f_i(s) \neq f_j(s)\} \tag{5.20}$$

Ces fonctions, dont le nombre est $|c|!$, représentent toutes les permutations possibles de l'ensemble c. Quelles que soient deux fonctions $f_i, f_j \in F_c$ l'output de l'une peut être utilisé comme input de l'autre, le résultat étant une composition de fonction : $(f_i \circ f_j)(s) \triangleq f_i(f_j(s))$. A partir de cette composition, nous pouvons considérer un groupe de symétrie [BUT 1991].

Proposition 5.3.4. *Le groupe $G = (F_c, \circ)$, dont l'opération est la composition de fonction, est un groupe de symétrie.*

Démonstration. Clôture: il est clair que n'importe quelle application successive de n'importe quelle fonction $f \in F_c$ est une application injective de l'ensemble c vers l'ensemble c. Puisque F_c contient toutes les fonctions injectives pour l'ensemble c, alors $f_i \circ f_j = f_k \in F_c$.

Associativité : en utilisant la définition de la composition de fonctions, $\forall f_i, f_j, f_k \in F_c, ((f_i \circ f_j) \circ f_k)(s) = (f_i(f_j(f_k(s))))$. D'un autre côté, $(f_i \circ (f_j \circ f_k))(s) = (f_i(f_j(f_k(s))))$.

Elément neutre : soit $f_0(s) = s, \forall s \in c$, il est clair que $f_0 \in F_c$. Nous avons maintenant $(f_0 \circ f_i)(s) = f_0(f_i(s)) = f_i(s)$ et, d'un autre côté, $(f_i \circ f_0)(s) = f_i(f_0(s)) = f_i(s)$.

Symétrie : soit $f_i(s) = z, \forall f_i \in F_c$, nous définissons $f_i^{-1}(z) = s$. Alors, $(f_i \circ f_i^{-1})(z) = f_i(f_i^{-1}(z)) = z = f_0(z)$ et, d'un autre côté, $(f_i^{-1} \circ f_i)(s) = f_i^{-1}(f_i(s)) = s = f_0(s)$. Puisque $f_i^{-1}(z)$ est une fonction injective sur c, alors $f_i^{-1}(z) \in F_c$. □

5.3.3 Catsets et USL

A partir de la définition 5.2.7, un catset contient des catégories distinctes de la même couche. Puisque le langage L_{IEML} est fini, le nombre de catégories distinctes de la même couche est donné par le cardinal de l'ensemble C_L. Nous pouvons considérer C_L comme un groupe et plus particulièrement comme un *anneau* [BUT 1991] :

Proposition 5.3.5. *Le groupe $G = (C_L, \oplus, \otimes)$ avec l'opération ensembliste différence symétrique (Δ) et l'opération ensembliste intersection ($\cap$), est un anneau.*

Démonstration. Le résultat de la différence symétrique comprend les membres qui sont dans l'un des deux ensembles, mais pas dans les deux : pour les ensembles A et B, cela équivaut à $(A \backslash B) \cup (B \backslash A)$. Dès lors, $\forall A, B, C \subseteq C_L$ le groupe $(C_L, \oplus)$ est un *groupe Abélien* puisque $\forall c_i, c_j \in C_L,\ c_i \oplus c_j \in C_L$ par la définition de l'ensemble des parties (clôture), $(A \oplus B) \oplus C = A \oplus (B \oplus C)$ par l'associativité de l'opération union ensembliste (associativité), $A \oplus B = B \oplus A$ par la commutativité de l'opération union ensembliste (commutativité),

$\exists c_0 \in C_L,\ \forall c_i \in C_L,\ c_i \oplus c_0 = c_0 \oplus c_i = c_i$ où $c_0 = \emptyset$ (élément neutre), $\exists c_i^{-1} \in C_L,\ \forall c_i \in C_L,\ c_i \oplus c_i^{-1} = c_i^{-1} \oplus c_i = c_0$ où $c_i^{-1} = c_i$ (symétrie) .

Le groupe $(C_L, \otimes)$ est un *monoïde* puisque $\forall c_i, c_j \in C_L,\ c_i \otimes c_j \in C_L$ par la définition de l'ensemble des parties (clôture), $(A \otimes B) \otimes C = A \otimes (B \otimes C)$ par l'associativité de l'opération ensembliste intersection (associativité), $\exists c_0 \in C_L,\ \forall c_i \in C_L,\ c_i \otimes c_0 = c_0 \otimes c_i = c_i$ où $c_0 = \emptyset$ (identité).

La multiplication se distribue sur l'opération d'addition : $\forall c_i, c_j, c_k \in C_L, c_i \otimes (c_j \oplus c_k) = (c_i \otimes c_j) \oplus (c_i \otimes c_k)$ et $(c_i \oplus c_j) \otimes c_k = (c_i \otimes c_k) \oplus (c_j \otimes c_k)$ par distributivité de l'intersection sur la différence symétrique. □

A partir de la définition 5.2.8, un USL est un ensemble de catsets de différentes couches. Puisqu'à chaque couche L il y a $|C_L|$ catsets distincts, la totalité de l'espace sémantique est défini par le produit :

$$\mathcal{USL} = C_0 \times C_1 \times C_2 \times C_3 \times C_4 \times C_5 \times C_6 \tag{5.21}$$

En considérant $\mathcal{USL}$ comme un groupe, on obtient la proposition suivante :

Proposition 5.3.6. *Le groupe $G_{\mathcal{USL}} = (\mathcal{USL}, \oplus, \otimes)$ avec les opérations différence symétrique (Δ) et intersection $(\cap)$ appliquées entre catégories de même couche, est un anneau [BUT 1991].*

Démonstration. La preuve est la même que pour la proposition 5.3.5. □

5.3.4 Symétries de transformation

Nous décrivons des symétries de transformation en utilisant le concept mathématique de Catégorie[5] [ARZ 1999],

$$\mathbb{C} = (O, M, \circ) \tag{5.22}$$

où O est une collection d'objets, M une collection de morphismes entre deux membres de la collection d'objets, et $\circ$ une opération de composition binaire entre morphismes compatibles. La Catégorie $\mathbb{C}$ a les propriétés suivantes : $\forall a, b, c \in O$, si $u = (a \to b) \in M$ et $v = (b \to c) \in M$ alors $\exists w = u \circ v = (a \to c) \in M$; $\forall u, v, w \in M$, si $(u \circ v) \circ w \in M$ alors $(u \circ v) \circ w = u \circ (v \circ w) \in M$; $\forall a \in O,\ \exists i_a \in M$ tel que $\forall u = (a \to b),\ i_a \circ u = u \ \wedge\ u \circ i_a = u$.

5.3.4.1 Le langage IEML considéré comme une Catégorie

Si nous considérons que les objets de la collection O sont des sous-ensembles du langage L_{IEML} à partir de l'équation 5.2 par exemple, $O = \{o_i \mid o_i \subset L_{IEML}; \forall s \in o_i,\ |s| = 3^i;\ 0 \leq i \leq 6\}$ alors les morphismes sont donnés par $M = \{m_i \mid m_i = (o_i \to o_{i+1});\ 0 \leq i \leq 5\}$. La fonction spécifique représentée par le morphisme est la fonction totale de *triplication*, $f_t : o_i \to o_{i+1}$ définie dans l'équation 5.3.

5. Il faut bien distinguer le concept mathématique de Catégorie (distingué ici par une capitale en initiale) du concept de catégorie (ensemble de séquences de même couche) en IEML.

5.3.4.2 Symétrie de permutation de rôle

Dans l'équation 5.22, la collection M contient aussi un automorphisme $m_r = (o_i \rightarrow o_i)$ représenté par une opération de permutation unaire agissant sur les rôles des sèmes d'une catégorie (voir la sous-section 5.2.3)

5.3.4.3 Symétrie de permutation de sème

Dans l'équation 5.22, la collection M contient encore un automorphisme $m_s = (o_i \rightarrow o_i)$ représenté par une opération de permutation binaire agissant sur les sèmes des catégories jouant un rôle donné (voir la sous-section 5.2.3)

5.4 La calculabilité des opérations sémantiques

5.4.1 Généralités

Pour démontrer la calculabilité des transformations sémantiques $f : S \rightarrow S$ où S représente les variables sémantiques (catégories, catsets ou USL), nous allons utiliser le formalisme des machines à états finis. Toutes les opérations résultant de transformations exécutées sur les variables sémantiques *sont* calculables selon les propriétés des machines à états finies.

5.4.1.1 Machines à états finies

Une machine à états finie (MEF) peut être définie [GIL 1962] comme suit :

Definition 5.4.1. *(MEF) Une machine à états finie M est un système synchrone représenté par un quintuple $M = (\Sigma, \Gamma, Q, \delta, \omega)$ avec un alphabet d'entrée fini $\Sigma = \{\sigma_1, \ldots, \sigma_i\}$, un alphabet de sortie fini $\Gamma = \{\gamma_1, \ldots, \gamma_j\}$, un ensemble fini d'états $Q = \{q_1, \ldots, q_n\}$ et une paire de fonctions caractéristiques δ et ω données par $q_{v+1} = \delta(q_v, \sigma_v)$, $\gamma_v = \omega(q_v, \sigma_v)$ où σ_v, γ_v, q_v sont respectivement le symbole d'entrée, le symbole de sortie et l'état de M à $v = 1, 2, \ldots$*

Ainsi, l'entrée (input) de la machine est une séquence de symboles $\sigma_1 \ldots \sigma_k$, et la sortie (output) de la machine est une séquence de symboles $\gamma_1 \ldots \gamma_k$.

En général, les machines peuvent changer d'état même s'il n'y a pas d'entrée, elles peuvent avoir plusieurs états de départ et il peut n'y avoir aucune, une ou plusieurs règles de transition exécutables. Pour accommoder ces différences, une fonction de transition est définie par $\delta_n : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$, où la valeur retournée est représentée par un ensemble des parties de Q. La fonction d'output peut être définie de la même manière par $\omega_n : Q \times \Sigma^* \times Q \rightarrow \Gamma^*$. Dans ces cas non déterministes, une notation alternative convient mieux. Au lieu des fonctions δ_n et ω_n, on utilise la relation de transition Δ :

Definition 5.4.2. *(NMEF) Une machine à états finie non déterministe M est un sextuple $M = (\Sigma, \Gamma, Q, Q_0, \Delta, F)$ où Σ est un alphabet d'entrée fini, Γ est un alphabet de sortie fini, Q est un ensemble fini d'états, $Q_0 \subseteq Q$ sont les états initiaux de la machine, $\Delta \subset Q \times \Sigma^* \times \Gamma^* \times Q$ est la relation de transition et*

Condition		Résultat	
Etat i	Entrée	Etat $i+1$	Sortie
q_0	-	q_2	1
q_1	-	q_0	0
q_2	0	q_3	0
q_2	1	q_1	0
q_3	0	q_0	0
q_3	1	q_1	0

TABLE 5.1 – Représentation d'une machine par une table de transition

F représente la condition d'acceptation (les états terminaux) étant précisé que $F \subseteq Q$.

Chaque relation de transition $(p, \sigma, \gamma, q) \in \Delta$, ou arête de l'état p à l'état q, peut être représentée par $p \xrightarrow{\sigma|\gamma} q$, où σ et γ jouent respectivement le rôle d'étiquette d'entrée et de sortie de l'arête. Etant donné un input et un état initial, la machine décrira un *chemin* qui est une séquence finie [BEA 2002] de relations représentées par : $q_0 \xrightarrow{\sigma_1|\gamma_1} q_1 \xrightarrow{\sigma_2|\gamma_2} q_2 \cdots \xrightarrow{\sigma_n|\gamma_n} q_n$.

Les étiquettes d'entrée et de sortie du chemin sont respectivement $\sigma_{in} = \sigma_1\sigma_2 \ldots \sigma_n$ et $\gamma_{out} = \gamma_1\gamma_2 \ldots \gamma_n$. La machine calcule alors la relation $(\sigma_{in}, \gamma_{out})$ sur $\Sigma^* \times \Gamma^*$. Un chemin est appelé *réussi* s'il part d'un état initial $q_s \in Q_0$ et finit dans un état q_f qui remplit une condition d'acceptation (un état terminal), habituellement défini par $q_f \in F \subseteq Q$. On voit donc qu'en général, et par contraste avec les machines déterministes, une machine non déterministe peut avoir plusieurs chemins réussis pour la même étiquette d'entrée σ_{in} ce qui génère un ensemble de relations $R = \{(\sigma_{in}, \gamma_{out}), (\sigma_{in}, \kappa_{out}) \ldots (\sigma_{in}, \eta_{out})\}$.

La description d'une machine à états finis est donnée d'habitude par une table de transition (table 5.1) ou par un graphe de transition (figure 5.1) [HOL 1991] à partir desquels on peut obtenir les fonctions δ et ω. Par exemple, la même machine peut être représentée par une table de transition où les rangées sont un sous-ensemble des relations $(Q \times \Sigma \times Q \times \Gamma)$, ou par un graphe de transition où les arêtes sont étiquetées par un sous-ensemble des relations $(\Sigma \times \Gamma)$.

5.4.1.2 Automates finis

L'automate déterministe $A_D = (\Sigma, Q, q_0, \delta, F)$ est une forme particulière de la machine à états finie. L'équation ci-dessous décrit la fonction de transition étendue $\hat{\delta}$ qui retourne un état p quand elle part de n'importe quel état q étant donnée une séquence d'input valide quelconque w :

$$\hat{\delta}(q, w) = p \tag{5.23}$$

Cette fonction, où $q \in Q$ et $w \in \Sigma^*$, est définie pour tous les automates. Ceci peut être démontré par induction sur la longueur de la chaîne d'input.

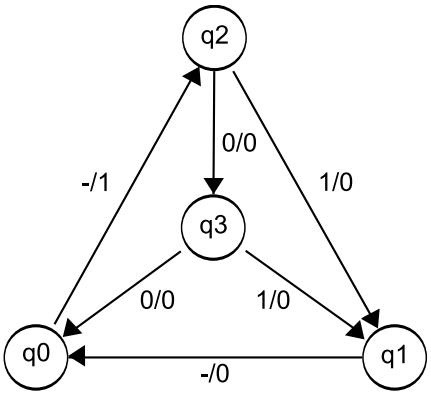

FIGURE 5.1 – Représentation d'une machine par un graphe

Base : on pose $q = \hat{\delta}(q, \epsilon)$, ce qui nous assure qu'il n'y a pas d'état de transition quand il n'y a pas d'input;

Induction : on représente l'ensemble w comme ua, où a est le dernier symbole et u est la chaîne originale sans le dernier symbole. Alors,

$$\hat{\delta}(q, ua) = \delta(\hat{\delta}(q, u), a) \tag{5.24}$$

Un langage [6] L de A_D dénoté $L(A_D)$ [HOP 2001] est alors défini par l'équation qui suit :

$$L(A_D) = \{w \mid \hat{\delta}(q_0, w) \in F\} \tag{5.25}$$

Nous pouvons définir l'état d'équivalence [7] dans les termes de l'ensemble F et de l'équation 5.23 comme suit. Deux états $p, q \in Q$ sont équivalents si et seulement si $\forall s \in \Sigma^*$, $\hat{\delta}(p, s) \in F \ \wedge \ \hat{\delta}(q, s) \in F$. En général, deux automates sont équivalents si et seulement si $L(A_i) = L(A_j)$, c'est-à-dire $L(A_i) \subseteq L(A_j)$ et $L(A_j) \subseteq L(A_i)$. Pour anticiper sur la suite, cette équivalence d'état nous fournit une méthode pour déterminer si des USL sont équivalents.

Les langages acceptés par les automates à états finis sont appelés des langages réguliers. Etant donnés deux langages $L(A_i)$ et $L(A_j)$, alors les opérations montrées dans la Table 5.2 ont pour résultat d'autres langages réguliers acceptés par un automate A_k [HEN 1968]. L'étoile de Kleene (ou fermeture itérative) d'un langage représente l'ensemble des chaînes qui peuvent être obtenues en

6. Une *expression régulière* peut aussi être utilisée pour spécifier tous les langages réguliers. Il existe des procédures algorithmiques pour convertir une expression régulière en automate à états finis et *vice versa*.

7. Si R est une relation sur $S \times S$, alors...
- R est réflexif si $(a, a) \in R \ \forall \ a \in S$,
- R est symétrique si $(b, a) \in R$ alors $(a, b) \in R$
- R est transitif si $(a, b) \in R$ et $(b, c) \in R$ alors $(a, c) \in R$.

Une relation d'équivalence ($\equiv$) est réflexive, symétrique et transitive. Les relations d'équivalence donnent lieu à des classes d'équivalence qui contiennent tous les membres concernés et eux seulement.

Opération	*Représentation*
Union	$L(A_k) = L(A_i) \cup L(A_j)$
Intersection	$L(A_k) = L(A_i) \cap L(A_j)$
Différence	$L(A_k) = L(A_i) - L(A_j)$
Complément	$L(A_k) = \Sigma^* - L(A_j)$
Concaténation	$L(A_k) = L(A_i)L(A_j)$
Etoile de Kleene	$L(A_k) = L(A_i)^*$

TABLE 5.2 – Opérations dont les résultats sont des langages réguliers

prenant n'importe quel nombre de chaînes de symboles de ce langage et en les concaténant.

5.4.1.3 Transducteurs à états finis

Selon le problème à résoudre, les transducteurs à états finis peuvent être interprétés de plusieurs manières [ROC 1997].

On peut voir les transducteurs à états finis comme des automates à états finis avec un alphabet $\Sigma = \Sigma_1 \times \Sigma_2$, acceptant ou rejetant des chaînes de paires (a, b) où $a \in \Sigma_1^*$ et $b \in \Sigma_2^*$. Cet automate est donné par $A = (\Sigma, Q, q_0, \delta, F)$ où $F \subseteq Q$ représente une condition d'acceptation cohérente avec le problème que l'on veut résoudre et $\delta = \{(p, (a, b), q) \mid (p, a, b, q) \in Q \times \Sigma_1^* \times \Sigma_2^* \times Q\}$ [8].

On peut aussi considérer les transducteurs à états finis comme des traducteurs quand on considère une classe d'applications de chaînes définies sur Σ^* vers des ensembles de chaînes $\mathcal{P}(\Gamma^*)$. Cette application est rationnelle si elle peut être réalisée par un transducteur à états finis. En supposant que $\delta(q, \sigma)$ puisse retourner un ensemble d'états et en utilisant

$$\hat{\omega}(q, s) = \hat{\omega}(q, \sigma s') = \omega(q, \sigma)\hat{\omega}(\delta(q, \sigma), s') = s'' \tag{5.26}$$

nous pouvons définir une *transduction rationnelle* $t : \Sigma^* \to \mathcal{P}(\Gamma^*)$ telle que $t(s_i) = \{s_o \mid \exists\, \hat{\omega}(q_0, s_i), s_i \in \Sigma^*\}$ et une *fonction rationnelle* si $|t(s_i)| \leq 1, \forall s_i \in \Sigma^*$ ($|t(s)|$ représente le nombre d'applications pour un input s.)

Finalement, un transducteur à états finis peut être vu comme calculant des relations entre des ensembles de chaînes [9]. Par analogie avec les automates à états finis qui n'acceptent de langages que si et seulement si ces langages sont réguliers, les transducteurs n'acceptent (ne calculent) des relations que si et seulement si ces relations sont régulières.

Les relations régulières sont définies comme [KAP 1994]: $\{\emptyset\}$ et $\{(a_i, a_j) \mid a_i, a_j \in A\}$, où A est un automate et a sont des chaînes. De plus, si R_a, R_b et R_c sont des relations régulières, alors les équations ci-dessous décrivent aussi des relations régulières :

8. Ceci peut aussi être vu comme une classe de graphes orientés où les états sont les sommets et les arêtes étiquetées sont données par $E \subseteq Q \times \Sigma_1^* \times \Sigma_2^* \times Q$

9. Chaque sous-ensemble du produit cartésien (incluant l'ensemble vide) entre deux ensembles, $A \times B = \{(a, b) \mid a \in A, b \in B\}$, est une relation binaire sur $A \times B$

opération	*Représentation*
Union	$R(M_k) = R(M_i) \cup R(M_j)$
Complement	Généralement non clos
Différence	Généralement non clos
Concaténation	$R(M_k) = R(M_i)R(M_j)$
Etoile de Kleene	$R(M_k) = R(M_i)^*$
Intersection	Généralement non clos
Produit	$R(M_k) = L(A_i) \times L(A_j)$
Composition	$R(M_k) = R(M_i) \circ R(M_j)$

TABLE 5.3 – Opérations dont les résultats sont des relations régulières

1. $R_a \cdot R_b = \{(a_i b_n, a_j b_m) \mid (a_i, a_j) \in R_a \ \wedge \ (b_n, b_m) \in R_b\}$
2. $R_a \cup R_b = \{(r_i, r_j) \mid (r_i, r_j) \in R_a \ \vee \ (r_i, r_j) \in R_b\}$
3. $\cup_{k=0}^{\infty} R_c^k = \emptyset \cup R_c \cup R_c \cdot R_c \cup R_c \cdot R_c \cdot R_c \ldots$

Les R^k représentent k concaténations de R. Donc, par définition les relations régulières sont closes par rapport aux opérations de concaténation, d'union et d'étoile de Kleene. Les opérations sur les relations régulières ayant pour résultat d'autres relations régulières sont présentées dans la Table 5.3. Par contraste avec l'opération intersection sur les langages réguliers, les relations régulières ne sont généralement pas closes par rapport à l'intersection. Pour utiliser un exemple souvent cité, deux relations régulières [10] $R_1 = (a^n b^*, c^n)$ et $R_2 = (a^* b^n, c^n)$ mettent en relation un langage régulier avec un autre langage régulier, tandis que $R_1 \cap R_2 = (a^n b^n, c^n)$ met en relation un langage non-régulier avec un langage régulier.

Si nous utilisons l'output d'une machine comme l'input d'une autre, les machines sont alors connectées en série (ou en cascade). Il est possible de combiner les différentes machines individuelles en une machine équivalente par l'opération de composition. En utilisant les définitions pour les machines à états finies $M_i = (\Sigma, H, Q_i, \delta_i, \omega_i)$ et $M_j = (H, \Gamma, Q_j, \delta_j, \omega_j)$, nous obtenons $M_k = (\Sigma, \Gamma, Q_i \times Q_j, \delta_k, \omega_k)$ où

$$\begin{aligned} \delta_k((p,q),\sigma) &= \{(p^{'}, q^{'}) \mid \delta_i(\sigma, p) = p^{'}, \delta_j(\omega_i(\sigma, p), q) = q^{'}\} \\ \omega_k((p,q),\sigma) &= \{\sigma' \mid \sigma' = \omega_j(\omega_i(\sigma, p), q)\} \end{aligned} \tag{5.27}$$

Les chaînes d'input $L_I(M) = \{s \in \Sigma^* \mid \exists(s, z) \in R(M)\}$ et les chaînes d'output $L_O(M) = \{z \in \Gamma^* \mid \exists(s, z) \in R(M)\}$ (et donc l'automate d'input et d'output) peuvent être retrouvées à partir du transducteur M par des opérations de projection.

Nous pouvons aussi combiner des machines en parallèle. En utilisant les définitions des machines à états finis $M_i = (\Sigma, \Gamma, Q_i, \delta_i, \omega_i)$ et $M_j = (\Sigma, \Gamma, Q_j, \delta_j, \omega_j)$,

10. Dans l'équation qui suit le n en exposant note le nombre de répétitions d'un symbole donné, et $*$ note une répétition infinie d'un symbole donné.

nous obtenons $M_k = (\Sigma, \Gamma, Q_i \times Q_j, \delta_k, \omega_k)$ où,

$$\begin{aligned}\delta_k((p,q),\sigma) &= \{(p^{'}, q^{'}) \mid \delta_i(p,\sigma) = p^{'} \ \wedge \ \delta_j(q,\sigma) = q^{'} \ \wedge \ \exists\, \omega_k((p,q),\sigma)\} \\ \omega_k((p,q),\sigma) &= \{\sigma' \mid \omega_i(p,\sigma) = \sigma' \ \wedge \ \omega_j(q,\sigma) = \sigma'\}\end{aligned} \tag{5.28}$$

La connexion en parallèle est associative et commutative. Elle est seulement définie au cas où toutes les machines sont ϵ-free.[11] En pratique, le nombre d'états résultant est inférieur au produit du nombre d'états des machines qui composent le montage en parallèle puisque beaucoup de ces états sont non-définis ou inatteignables.

5.4.2 Transformations de catégories

Le but de cette sous-section est de montrer que chaque catégorie peut être transformée en une autre catégorie, (y compris elle-même). Nos entrées et nos sorties sont des langages réguliers (l'ensemble de toutes les chaînes reconnues par un automate à états finis). L'espace de toutes les combinaisons possibles est constitué par l'ensemble des appariements entre chacune des entrées possibles et chacune des sorties possibles. Si l'on présente à une machine n'importe quelle paire de chaînes d'entre et de sortie et que cette machine finit par s'arêter dans un état terminal (état d'acceptation), alors nous avons montré que n'importe quelle catégorie peut effectivement être transformée en n'importe quelle autre catégorie.

La preuve que la transformation d'une catégorie dans une autre est calculable s'appuie sur la construction d'un transducteur approprié. Pour montrer que l'on peut créer un transducteur capable de reconnaître les relations $L(A_1) \times L(A_2)$, nous utilisons le raisonnement suivant : pour commencer, nous utilisons les définitions de $L(A_1)$ et $L(A_2)$ pour définir le transducteur T. Nous montrons alors que ce transducteur a la propriété d'apparier chaque mot de $L(A_1)$ à un mot de $L(A_2)$. La chose est réalisée en montrant que T atteint un état terminal si et seulement si les deux automates représentant $L(A_1)$ et $L(A_2)$ atteignent aussi des états finaux. Nous en concluons qu'il existe un transducteur T qui calcule n'importe quelle paire input/output valide, à moins que $L(A_1)$ ou $L(A_2)$ ne soient eux-même mal définis.

Theorem 5.4.1. *Etant donnée une catégorie quelconque $c \neq \emptyset$, il existe un transducteur à états fini T qui l'applique à un sous-ensemble de C_L: $\forall c \in C_L \ \exists T \mid c' = T(c),\ c' \subseteq C_L$. La calculabilité de cette application suit directement de l'existence du transducteur T.*

Démonstration. Nous représentons les catégories comme des langages (équation 5.25) et nous avons besoin de montrer que les relations $L(A_1) \times L(A_2)$ sont reconnues par un transducteur T. Dans ce cas, $L(A_1)$ est le langage de l'automate $A_1 = (\Sigma, Q_1, q_1, F_1, \delta_1)$ qui représente la catégorie c et $L(A_2)$ est le langage de

11. Dans les machines qualifiée d'ϵ-free, toutes les transitions sont étiquetées par un symbole, ce qui exclut les transitions fallacieuses.

l'automate $A_2 = (\Sigma, Q_2, q_2, F_2, \delta_2)$ qui représente la catégorie c'. Le transducteur est ainsi réglé : $T = (\Sigma \cup \{\epsilon\}, Q_1 \times Q_2, (q_1, q_2), F_1 \times F_2, \delta_T)$. La fonction de transition δ_T est définie comme :

$$\delta_T((q_1, q_2), a, b) = \delta_1(q_1, a) \times \delta_2(q_2, b) \tag{5.29}$$

où $q_1 \in Q_1; q_2 \in Q_2; a, b \in \Sigma \cup \{\epsilon\}$. Il nous faut montrer que pour chaque paire (u, v) où $u, v \in \Sigma^*$, T se trouve dans un état terminal $(f_n, f_m) \in F_1 \times F_2$ si et seulement si A_1 et A_2 sont respectivement dans les états terminaux f_n et f_m . A cette fin, nous devons montrer la validité de l'équation suivante :

$$\hat{\delta}_T((q_1, q_2), x, y) = \hat{\delta}_1(q_1, x) \times \hat{\delta}_2(q_2, y) \tag{5.30}$$

Par induction sur le nombre de transitions (ou sur la longueur de la chaîne d'input si le transducteur est déterministe) nous avons :

Base: $\hat{\delta}_T((q_1, q_2), \epsilon, \epsilon) = \delta_T((q_1, q_2), \epsilon, \epsilon) = \delta_1(q_1, \epsilon) \times \delta_2(q_2, \epsilon) = \hat{\delta}_1(q_1, \epsilon) \times \hat{\delta}_2(q_2, \epsilon)$;

Induction: $\hat{\delta}_T((q_1, q_2), ua, vb) = \delta_T(\hat{\delta}_T((q_1, q_2), u, v), a, b)$ en utilisant l'équation 5.24; $\delta_T(\hat{\delta}_T((q_1, q_2), u, v), a, b) = \delta_T(\hat{\delta}_1(q_1, u) \times \hat{\delta}_2(q_2, v), a, b)$ est le pas inductif de l'équation 5.30; $\delta_T(\hat{\delta}_1(q_1, u) \times \hat{\delta}_2(q_2, v), a, b) = \delta_1(\hat{\delta}_1(q_1, u), a) \times \delta_2(\hat{\delta}_2(q_2, v), b)$ en utilisant l'équation 5.29; et finalement, $\delta_1(\hat{\delta}_1(q_1, u), a) \times \delta_2(\hat{\delta}_2(q_2, v), b) = \hat{\delta}_1(q_1, ua) \times \hat{\delta}_2(q_2, vb)$ en utilisant l'équation 5.24. Le transducteur T est donc dans un état terminal (état d'acceptation) si et seulement si aussi bien A_1 que A_2 sont également dans des états terminaux. □

5.4.3 Transformations de catsets et d'USL

Puisque les catsets sont des ensembles de catégories et que les USL sont des ensembles de catsets, nous pouvons utiliser le Théorème 5.4.1 pour montrer que les transformations de catsets et d'USL sont, elles aussi, calculables.

Theorem 5.4.2. *Étant donné un catset quelconque $\kappa_i \neq \emptyset$, il existe un transducteur à états fini T qui applique κ_i à un catset κ_o. La calculabilité de cette application suit directement de l'existence du transducteur T.*

Démonstration. La transformation d'un catset κ_i en un catset κ_o implique que chaque *catégorie* du catset $c_i \in \kappa_i$ soit transformée en une *catégorie* $c_o \in \kappa_o$. A la fois c_i et c_o représentent des langages, et nous pouvons représenter les catsets en termes de langages : $L_\kappa = \bigcup_{n=0,\ c_n \in \kappa}^{m} c_n$. On établit que L_{κ_i} et L_{κ_o} sont les langages des catsets κ_i et κ_o respectivement, où aussi bien L_{κ_i} que L_{κ_o} sont des sous-ensembles de L_{IEML} par l'équation 5.9, nous obtenons d'abord deux automates qui reconnaissent L_{κ_i} et L_{κ_o} et nous suivons ensuite le même raisonnement que dans le Théorème 5.4.1 pour montrer l'existence d'un transducteur T qui encode la relation entre L_{κ_i} et L_{κ_o}. □

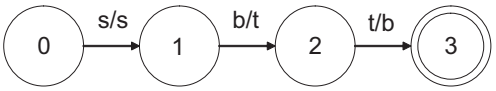

FIGURE 5.2 – Opération d'échange de sème (*attribut* ⇔ *mode*) pour la *catégorie* {*sbt*}

Theorem 5.4.3. *Etant donné un USL $u_i \neq \emptyset$, il existe un transducteur à états fini T qui applique u_i à un USL u_o. La calculabilité de cette application suit directement de l'existence du transducteur T.*

Démonstration. Pour montrer l'existence d'un transducteur T qui réalise l'application $u_o = T(u_i)$, on utilise le Théorème 5.4.2 : puisqu'un transducteur qui effectue l'application $\kappa_o = t(\kappa_i)$ existe pour chaque $\kappa_i \in u_i$ et $\kappa_o \in u_o$, et que les transducteurs sont clos eu égard à l'opération union (voir la table 5.3), alors $T = \bigcup_{n=0}^{6} t_n$, où t_n est le transducteur qui effectue l'application $\kappa_o^n = t_n(\kappa_i^n)$. □

5.4.4 Exemples d'opérations sur des catégories

Cette sous-section porte sur certaines opérations sur les catégories IEML. Elle se fonde sur un résultat déjà obtenu, le Théorème 5.4.1. Bien que ce théorème nous assure qu'il existe une machine pour une opération particulière quelconque, il ne donne pas le détail de la manière de créer cette machine. Les exemples qui suivent montrent des machines capables d'accomplir une opération donnée.

5.4.4.1 L'opération *échange de sèmes*

La machine qui suit exécute un simple échange de sèmes sur une catégorie :

$$\begin{aligned} &\Sigma = \Gamma = \{s, b, t\} \\ &Q = \{0, 1, 2, 3\} \\ &Q_0 = \{0\} \\ &\Delta = \{(0, s, s, 1), (1, b, t, 2), (2, t, b, 3)\} \\ &F = \{3\} \end{aligned} \tag{5.31}$$

La machine définie par l'équation 5.31 prend comme entrée une catégorie {*sbt*} et donne en sortie la catégorie {*stb*}. Cette machine est montrée dans la figure 5.2.

5.4.4.2 L'opération *powerset*

Considérant la fonction Powerset d'une *catégorie*, nous pouvons aisément construire une machine à états finie pour la représenter en utilisant la définition

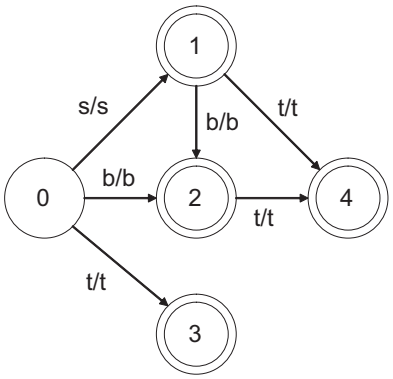

FIGURE 5.3 – Opération Powerset pour la *catégorie* $\{s, b, t\}$

5.4.2 :

$$\begin{aligned}
&\Sigma = \Gamma = \{s, b, t\} \\
&Q = \{0, 1, 2, 3, 4\} \\
&Q_0 = \{0\} \\
&\Delta = \{(0, s, s, 1), (0, b, b, 2), (0, t, t, 3), (1, b, b, 2), (1, t, t, 4), (2, t, t, 4)\} \\
&F = \{1, 2, 3, 4\}
\end{aligned} \tag{5.32}$$

La machine définie par l'équation 5.32 produit et reconnait l'ensemble suivant :$\{\{s\}, \{b\}, \{t\}, \{s, b\}, \{s, t\}, \{b, t\}, \{s, b, t\}\}$. Elle est représentée dans la figure 5.3. Dans cette figure, l'état 0 est l'état de départ et les doubles cercles représentent les états finaux - ou états d'acceptation. A titre d'exemple, si l'input de la machine est s alors que la machine est dans l'état de départ, la machine va produire s en sortie et va passer à l'état 1. Dans cet état, elle accepte seulement b et t ce qui fait passer la machine aux états 2 et 4 respectivement. A ce point la machine produit/reconnait l'ensemble suivant : $\{\{s\}, \{s, b\}, \{s, t\}\}$.

5.4.4.3 L'opération *partition*

L'opération partition requiert une opérande de base, une adresse de rôle et une opérande de partition. En termes mathématiques, la base est un ensemble S (possiblement un ensemble d'ensembles), l'adresse de rôle indique sur quelle partie $s \in S$ s'effectue la partition, et l'opérande de partition spécifie précisément la partition à exécuter. Cette opération applique des ensembles de couche l à des ensembles de couche l, $S^l \rightarrow P^l$ de telle sorte que $s \in S \iff s \in \bigcup_i p_i$ où $p_i \in P$ et $\bigcap_i p_i = \emptyset$.

La machine à états finie qui représente cette opération est un transducteur non-déterministe. Par exemple, une partition de la catégorie $f = \{u, a, s, b, t\}$ est l'ensemble $\{\{s\}, \{b\}, \{t\}, \{u\}, \{a\}\}$ qui est représenté par la machine suivante :

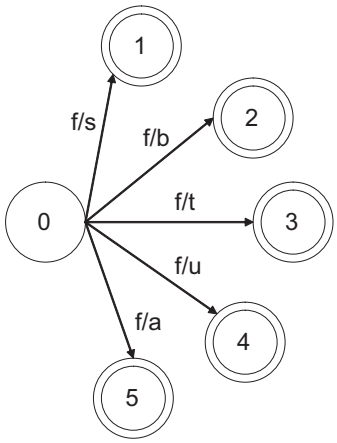

FIGURE 5.4 – Opération de partition pour la catégorie $f = \{u, a, s, b, t\}$

$$\begin{gathered} \Sigma = \{f\} \\ \Gamma = \{s, b, t, u, a\} \\ Q = \{0, 1, 2, 3, 4, 5\} \\ Q_0 = \{0\} \\ \Delta = \{(0, f, s, 1), (0, f, b, 2), (0, f, t, 3), (0, f, u, 4), (0, f, a, 5)\} \\ F = \{1, 2, 3, 4, 5\} \end{gathered} \tag{5.33}$$

La machine définie par l'équation 5.33 produit/reconnait l'ensemble suivant : $\{s, b, t, u, a\}$, comme on peut le voir sur la figure 5.4. Dans cette figure, l'état 0 est l'état de départ et les doubles cercles représentent les états finaux (ou états d'acceptation). Si l'input de la machine est F tandis que la machine est à l'état de départ, la machine va produire $\{s, b, t, u, a\}$ en sortie et va passer par les états $1, 2, 3, 4, 5$.

5.4.4.4 L'opération *rotation*

L'opération de rotation requiert une base, une adresse de rôle et un rotor. En termes mathématiques, la base est un ensemble S, l'adresse de rôle désigne sur quelle partie de $s \in S$ doit être effectuée la rotation, et le rotor précise la rotation à exécuter. Cette opération applique des ensembles de couche l à des ensembles de couche $l + 1$, $S^l \to P^l$.

La machine à états finie qui représente cette opération est un transducteur non-déterministe. Par exemple, si le rotor est donné par $\{\{s\}, \{b\}, \{t\}\}$ et si l'adresse de rôle est l'attribut, alors la rotation de la *catégorie* $\{sbt\}$ est l'ensemble $\{\{sst\}, \{sbt\}, \{stt\}\}$. La machine qui suit représente cette rotation.

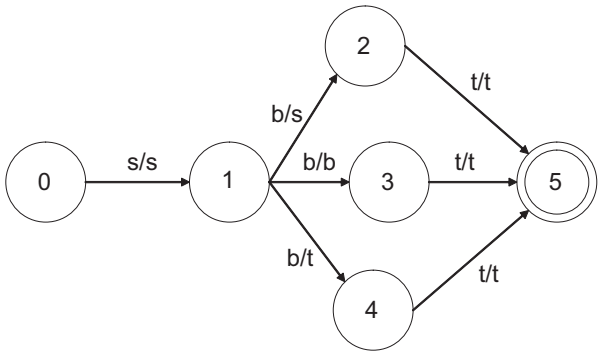

FIGURE 5.5 – Opération de Rotation pour la catégorie $\{sbt\}$ avec le rotor $\{\{s\},\{b\},\{t\}\}$ et l'adresse de rôle "attribut".

$$\begin{gathered}\Sigma = \Gamma = \{s, b, t\} \\ Q = \{0, 1, 2, 3, 4, 5\} \\ Q_0 = \{0\} \\ \Delta = \{(0, s, s, 1), (1, b, s, 2), (1, b, b, 3), (1, b, t, 4), (2, t, t, 5), (3, t, t, 5), (4, t, t, 5)\} \\ F = \{5\}\end{gathered} \tag{5.34}$$

Il faut noter que l'adresse de rôle va décider où le non-déterminisme va avoir lieu : dans l'exemple en question, il s'agit du second état. La machine définie par l'équation 5.36 produit/reconnait l'ensemble suivant : $\{sst, sbt, stt\}$, ainsi qu'on peut le voir sur la figure 5.5. Dans cette figure, l'état 0 est l'état de départ et le double cercle représente l'état terminal (l'état d'acceptation).

5.4.4.5 L'opération de supertriplication

L'opération de supertriplication est l'union de toutes les opérations de triplication possibles (voir l'équation 5.3). Une fois que des transducteurs pour les opérations de triplication sont définis, on effectue une opération d'union sur ces transducteurs, ce qui a pour résultat un autre transducteur (les transducteurs sont clos eu égard à l'opération d'union, comme nous l'avons vu Table 5.3) L'opération de supertriplication est donc aussi représentée par un transducteur (composite).

5.4.4.6 Opérations de supersélection

Pour une couche donnée, le fait d'imposer un ordre particulier sur des ensembles de séquences nous permet d'effectuer une opération de supersélection sur ces ensembles. En termes mathématiques, étant donné deux ensembles ordonnés A et B pour lesquels $|A| = |B|$, $A \times B \to C$, où $\forall c_{ij} \in C$, $a_i \in c_{ij} \wedge b_j \in c_{ij}$. L'opération de supersélection produit une matrice IEML régulière.

5.4.4.7 L'opération de concaténation de matrice

Les matrices peuvent être concaténées. Cette concaténation résulte de l'union des ensembles représentant les matrices. Il faut noter qu'il s'agit d'une opération distincte de celle qui consiste à réaliser l'union des ensembles sous-jacents. Supposons que $A \times B \to C$, $A^{'} \times B^{'} \to C^{'}$. Dans le premier cas nous avons $C \cup C^{'} = C^{''}$ tandis que dans le second cas nous obtiendrions $(A \cup A^{'}) \times (B \cup B^{'}) = C \cup C^{'} \cup A \times B^{'} \cup A^{'} \times B$.

5.5 Les relations sémantiques

5.5.1 Généralités

En IEML, les relations sont modélisées par des graphes. On peut définir un graphe par une paire $G = (V, E)$ d'ensembles, tel que E est un sous-ensemble de tous les sous-ensembles à deux éléments de V: $E \subseteq [V]^2$. A titre d'illustration, si $V = \{a, b, c, d\}$ alors $[V]^2 = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}$, E peut être représenté par exemple par $\{\{a, b\}, \{b, c\}\}$. En général, le nombre de membres de l'ensemble $[V]^k$ (sous-ensembles de V à k éléments) est une combinaison de n éléments choisis k fois sans répétition et qui peut être exprimé par $\frac{n!}{k!(n-k)!}$.

Si les sommets sont les catégories (au sens d'IEML) d'un ensemble donné, et si les arêtes représentent les relations entre ces catégories, le graphe résultant modélise le réseau de relations entre les catégories de l'ensemble. De même, si les sommets représentent des ensembles de catégories, et les arêtes représentent les relations entre ces ensembles, le graphe résultant modélise le réseau de relations entre les ensembles de catégories.

Le nombre de sommets d'un graphe est appelé son ordre et on le représente par $|G|$, tandis que le nombre d'arêtes est représenté par $||G||$. Les graphes peuvent être finis, infinis dénombrables ou infinis non dénombrables, selon leur ordre. Les arêtes connectent les sommets et deux sommets a, b sont dits adjacents si l'arête $ab \in G$.

5.5.1.1 Similarité et emboîtement des relations

Deux graphes sont isomorphes s'il existe une bijection $\phi : V \to V'$ telle que $ab \in E \iff \phi(a)\phi(b) \in E'$ pour tout $a, b \in V$. En somme, il doit exister une application entre les sommets de deux graphes tels que tous les sommets soient préservés. L'existence d'un isomorphisme entre deux relations signifie que la structure de l'une est semblable à la structure de l'autre.

Un sous-graphe G' de G est noté $G' \subseteq G$. La condition requise pour que $G' \subseteq G$ est que $V' \subseteq V$et que $E' \subseteq E$. Pour un sous-graphe G', s'il contient toutes les arêtes $ab \in E$ avec $a, b \in V'$ alors c'est un sous-graphe induit de G et il est noté $G' = G[V']$. Les sous-graphes représentent de manière efficace les emboîtements de relations.

5.5.1.2 Connectivité des relations

Le degré $d(v)$ d'un sommet v est le nombre d'arêtes à v. Le degré minimum et maximum (pas l'ordre!) d'un graphe est représenté par $\delta(G) \triangleq min\{d(v)|v \in V\}$ et $\Delta(G) \triangleq max\{d(v)|v \in V\}$ respectivement. Le degré moyen d'un graphe est donné par : $\frac{1}{|V|}\Sigma_{v \in V} d(v)$.

Si tous les sommets ont le même degré, alors le graphe est régulier (par exemple, un graphe cubique, pour le degré 3).

5.5.1.3 Relations hiérarchiques : arbres

Une relation peut être hiérarchique, elle sera alors représentée par un arbre. Un graphe G qui ne contient aucun cycle est une forêt. En ajoutant des arêtes à une forêt jusqu'à la rendre connexe, on obtient un arbre. Nous allons maintenant énoncer certaines propriétés générales des arbres, propriétés qui valent évidemment pour tous les types de relations sémantiques représentables par des arbres :

Proposition 5.5.1. *Etant donné un graphe G, s'il y a un chemin* unique *entre deux sommets quelconques, alors G est un arbre.*

Démonstration. Le graphe est connexe puisqu'il existe un chemin entre deux sommets quelconques. Un cycle requiert deux chemins entre les mêmes sommets. Un chemin unique entre deux sommets quelconques d'un graphe implique donc qu'il n'y a pas de cycles dans le graphe. □

Proposition 5.5.2. *Etant donné un arbre T, le chemin connectant deux sommets quelconques de T est unique.*

Démonstration. Tous les sommets sont connectés, et puisqu'il n'y a pas de cycles dans T, le chemin connectant deux sommets doit être unique. □

Proposition 5.5.3. *Si une nouvelle arête joint deux sommets d'un arbre T, alors un cycle est formé.*

Démonstration. Puisque T est un arbre, il existe un chemin unique de c à u et de c à v, $\forall\, c, u, v \in T$. Si une nouvelle arête joint u et v, un cycle $c \ldots uv \ldots c$ est formé. □

Proposition 5.5.4. *Un arbre T avec n sommets possède $n-1$ arêtes.*

Démonstration. (informelle) Un arbre avec un seul sommet n'a pas d'arête. En ajoutant un second sommet à l'arbre on ajoute exactement une arête : cela ne peut être moins, sinon le graphe ne serait pas connexe, et cela ne peut être plus, sinon un cycle serait créé. Le même raisonnement peut être fait pour l'ajout de chaque nouveau sommet. □

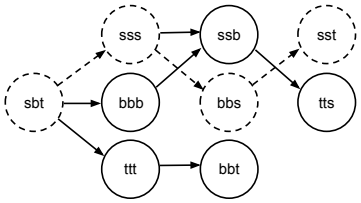

FIGURE 5.6 – Représentation graphique d'une relation. La portion en pointillé est un chemin entre *sbt* et *sst* et elle peut être vue comme une catégorie $\{sbt, sss, bbs, sst\}$.

5.5.1.4 Opérations de base sur les relations

L'union de deux graphes G et G' est, par définition, $G \cup G' \triangleq (V \cup V', E \cup E')$, c'est-à-dire, l'union respective des sommets et des arêtes des graphes. De même, l'intersection de deux graphes est, par définition, $G \cap G' \triangleq (V \cap V', E \cap E')$. La différence entre deux graphes est obtenue en éliminant tous les sommets communs et leurs arêtes incidentes. Il existe beaucoup d'autres opérations qui peuvent être utilisées pour les relations. En voici deux exemples.

1) Si G et G' sont des graphes disjoints (c'est-à-dire si $V \cap V' = \emptyset$), alors $G * G'$ est le graphe obtenu de $G \cup G'$ après avoir connecté tous les sommets de G à tous les sommets de G'.

2) Le complément de G, dénoté $\bar{G}$ est le graphe sur V avec $\bar{E} = [V]^2 \backslash E$.

5.5.1.5 Chemins relationnels

Chemins Un chemin est un graphe $P = (V, E)$ tel que $V = \{x_1, x_2, \ldots, x_n\}$ et $E = \{x_1x_2, x_2x_3, \ldots x_{n-1}x_n\}$. Un chemin peut alors être représenté comme une séquence de sommets, $x_1x_2 \ldots x_n$ de x_1 à x_n. Un graphe G est connexe si deux de ses sommets quelconques sont connectées par un chemin dans G. Les chemins peuvent représenter des catégories IEML (voir la figure 5.6) et fournissent une manière intéressante de décrire des catsets et des USL.

Cycles relationnels Etant donné un chemin $P = x_1x_2 \ldots x_{n-1}$ où $n \geqslant 3$ alors le graphe $C = P + x_{n-1}x_1$ est un cycle (deux arêtes connectant les mêmes sommets dans un graphe non orienté est traité comme une seule arête).

Les longueurs minimale et maximale des cycles contenus dans un graphe G sont appelés respectivement calibre et circonférence. Dans le cas d'un arbre (voir section 5.5.1.3), le calibre est considéré comme étant infini ∞ et la circonférence comme égale à 0.

Distances relationnelles La distance entre deux sommets x et y d'un graphe G, $d_G(x, y)$ est la longueur du chemin le plus court entre x et y. S'il n'existe aucun chemin, la distance est ∞. L'excentricité d'un sommet x est la distance maximale entre x et un autre sommet du graphe G. Le diamètre d'un graphe est

donné par l'excentricité maximale de ses sommets, $diam_G = max_{y \in V(G)}(d_G(x, y))$ et son rayon par l'excentricité minimale de ses sommets, $rad_G = min_{x \in V(G)}(diam_G)$.

Un sommet d'un graphe G dont la distance à n'importe quel autre sommet est inférieure ou égale au rayon du graphe est central dans ce graphe.

Rapport entre rayon, degré et nombre de sommets d'un graphe Étant donné un graphe représentant n'importe quel type de relation sémantique, son rayon et son degré minimal peuvent être utilisés à des fins de classification. Par exemple, un graphe G représentant n'importe quel type de relation sémantique, dont le rayon est au plus k et dont le degré $d \geq 3$ possède moins de $\frac{d}{d-2}(d-1)^k$ sommets.

Démonstration. Soit z un sommet central de G et D_i l'ensemble de sommets de G à distance i de z. Le nombre total de sommets de G est donné par $V(G) = \bigcup_{i=0}^{k} D_i$. On peut donc énoncer que :

- Pour $i = 0$, $|D_0| = 1$ puisqu'il ne contient que le sommet z.

- Pour $i = 1$, $|D_1| \leqslant d$ puisqu'il ne peut excéder le degré maximum du graphe.

- Pour $i \geqslant 1$, $|D_{i+1}| \leqslant (d-1)|D_i|$ puisque chaque sommet de D_{i+1} partage au moins une arête avec un sommet de D_i.

Nous obtenons $|D_{i+1}| \leqslant d(d-1)^i$, $\forall i < k$. Le nombre de sommets de G est donc $|V(G)| \leqslant \sum_{i=0}^{k} |D_i| = 1 + \sum_{i=0}^{k-1} |D_{i+1}|$. La somme $s_k = \sum_{i=0}^{k-1}(d-1)^i$ peut être réécrite comme $s_k = 1 + (d-1) + (d-1)^2 + \ldots + (d-1)^{k-1}$ et peut être soustraite de $s_k(d-1)$ ce qui donne $s_k(d-1) - s_k = (d-1)^k - 1$. La solution pour s_k donne $\frac{(d-1)^k - 1}{d-2}$. En replaçant cette solution dans l'équation précédente, nous obtenons $|V(G)| \leqslant 1 + d\frac{(d-1)^k - 1}{d-2} < \frac{d}{d-2}(d-1)^k$ puisque $\frac{d}{d-2} > 1$ □

5.5.2 Relations et graphes sémantiques

Les graphes sémantiques sont une représentation des relations sémantiques entre les expressions IEML : catégories, catsets et USL.

5.5.2.1 Relations d'ordre linéaires

N'importe quel graphe $G = (V, E)$ qui est un arbre (voir la sous-section 5.5.1.3) et pour lequel $\forall v \in V$, le degré (voir la sous-section 5.5.1.2) se conforme toujours à $0 < d(v) \leq 2$, décrit un chemin (voir la sous-section 5.5.1.5). Les arêtes E de ces graphes définissent une relation d'ordre linéaire entre catégories, catsets et USL. Il peut exister de nombreux graphes de ce type, dépendant des critères qui président à l'ordre linéaire.

5.5.2.2 Relations ensemble / sous-ensemble

Ce type de relation ne peut exister qu'entre catégories de même couche. Dans le cas général toutes les catégories qui sont dans une relation ensemble-sous-ensemble sont données par $C_L^r \subseteq C_L \times C_L$ où C_L^r est donné par :

$$C_L^r = \{\{c_i, c_j\} \mid c_i, c_j \in C_L, c_i \subseteq c_j\} \tag{5.35}$$

Le graphe $G = (V, E)$ est donné par :

$$\begin{aligned} V &= C_L \\ E &= C_L^r \end{aligned} \tag{5.36}$$

Les relations ensemble-sous-ensemble peuvent aussi être construites pour des cas particuliers en calculant l'ensemble des parties de la catégorie en question, ce qui définit les sommets V, et en appliquant la formule de l'équation 5.36. La figure 5.7 montre une représentation pour la *catégorie* $\{sbt, sbb, sss\}$.

Les relations ensemble-sous-ensemble sont aussi applicables aux catsets et aux USL : deux catsets différents sont dans une relation ensemble-sous-ensemble si et seulement si il existe une catégorie dans les deux catsets qui sont dans une relation ensemble-sous-ensemble; deux USL différents sont dans une relation ensemble-sous-ensemble si et seulement si il existe un catset dans chacun des deux USL qui sont dans une relation ensemble-sous-ensemble.

Dans le cas général, tous les catsets qui sont dans une relation ensemble-sous-ensemble sont donnés par $\kappa_L^r \subseteq C_L \times C_L$ où κ_L^r est donné par :

$$\kappa_L^r = \{\{\kappa, \kappa_j\} \mid \exists c_i \in \kappa_i, \exists c_j \in \kappa_j, \{c_i, c_j\} \in C_L^r\} \tag{5.37}$$

De la même manière, tous les USL qui sont dans une relation ensemble-sous-ensemble sont donnés par $\mathcal{USL}^r \subseteq \mathcal{USL} \times \mathcal{USL}$ où $\mathcal{USL}^r$ est donné par :

$$\mathcal{USL}^r = \{\{u_i, u_j\} \mid \exists \kappa_i \in u_i, \exists \kappa_j \in u_j, \{\kappa_i, \kappa_j\} \in \kappa_L^r\} \tag{5.38}$$

5.5.2.3 Relations symétriques

Ce type de relation se noue entre catégories c et c' de même couche et de même cardinalité (les catégories doivent contenir le même nombre de séquences), quand il existe un automate (voir la sous-section 5.4.1.1) A qui reconnaît une sous-séquence $s \in L_{IEML}$ identique dans c et c', et un transducteur (voir la sous-section 5.4.1.3) T qui calcule une relation cTc' (c'est-à-dire qui transforme c en c'). L'automate A décrit une similitude au niveau de l'arrangement symbolique (le niveau syntaxique) ce qui représente une condition nécessaire pour l'assertion d'une symétrie sémantique. Le transducteur T, quant à lui, décrit une invariance sémantique combinée à une variation sémantique. Les machines A et T peuvent être dérivées du *directory* (voir la section 5.6.4.5) contenant les catégories en question.

Des catsets sont en relation symétriques si et seulement si il existe une catégorie dans chacun des catsets qui sont en relation de symétrie. De même, des

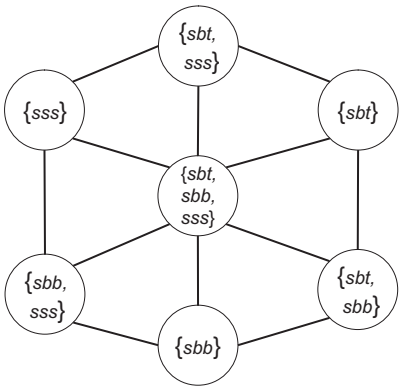

FIGURE 5.7 – représentation Graphique d'une relation ensemble-sous-ensemble pour la catégorie $\{sbt, sbb, sss\}$.

USL sont en relation de symétrie rela si et seulement si il existe un catset dans chacun des USL qui sont en relation de symétrie.

5.5.2.4 Relations étymologiques

Une relation étymologique est en général une relation entre une catégorie de couche n et l'un de ses sèmes de couche n–n' où $1 \leq n' < n$. Dépendant du directory qui contient les catégories, une relation étymologique peut être présente *ou non*. La représentation graphique de ces relations a une structure d'arbre.

5.6 Les circuits sémantiques

5.6.1 Caractères et séquences paradigmatiques

On distingue 10 symboles dans $\bar{\Sigma}$: $\{T\}$, $\{B\}$, $\{S\}$, $\{A\}$, $\{U\}$, $\{E\}$, $\{S, B, T, U, A, E\}$, $\{S, B, T, U, A\}$, $\{S, B, T\}$ and $\{U, A\}$ qui sont un sous-ensemble de l'alphabet du langage des catégories (voir l'équation 5.17). Ces symboles, et seulement ces symboles, sont appelés des caractères paradigmatiques et forment l'alphabet paradigmatique $\bar{\Sigma}_P$. L'usage de l'alphabet $\bar{\Sigma}_P$ nous permet de distinguer les séquences paradigmatiques parmi les séquences du langage des catégories (voir plus haut 5.2.4). Les séquences IEML sont obtenues à partir des séquences paradigmatiques en effectuant un produit cartésien entre tous les ensembles de la séquence paradigmatique. Par exemple, la séquence paradigmatique $\{\{S, B, T\}\{U, A\}\{E\}\}$ donne l'ensemble de séquences IEML $\{SUE, SAE, BUE, BAE, TUE, TAE\}$. En utilisant la définition de 5.1, les séquences paradigmatiques pour $0 \leq l \leq 6$ sont données par l'expression générale :

$$L^P_{IEML} = \{s \in \bar{\Sigma}^*_P \mid |s| = 3^l\} \tag{5.39}$$

dans laquelle $L^P_{IEML} \subseteq L_{IEML}$.

5.6.2 Distance paradigmatique

En utilisant l'équation 5.21 nous considérons un ensemble de fonctions $F = \{f_1, f_2, \cdots, f_n\}$ pour lesquelles

$$\forall f \in F,\ f : \mathcal{USL} \to \mathcal{USL} \tag{5.40}$$

et un graphe orienté $G_p = (\mathcal{USL}, E)$ où :

$$E = \{(\mu_i, \mu_o) \mid (\mu_i, \mu_o) \in [\mathcal{USL}]^2 \ \wedge\ \mu_o = f(\mu_i),\ f \in F\} \tag{5.41}$$

Le *chemin paradigmatique* entre deux noeuds quelconques $a, b \in \mathcal{USL}$ du graphe G_p est un chemin sur G_p (voir la sous-section 5.5.1.5), ce qui nous mène à la définition suivante d'une *distance paradigmatique* :

Definition 5.6.1. *Le plus court chemin entre deux sommets a, b de G_p est la distance paradigmatique entre a et b.*

Le plus court chemin peut être calculé par les algorithmes présentés dans la sous-section 5.7.2.

5.6.3 Circuits Sémantiques

Les circuits sémantiques S_ω sont des graphes orientés et étiquetés $G = (V, E)$ où $V \subseteq L_{IEML}$ et où tous les $e \in E$ représentent une relation ternaire (s, m, d) dans laquelle $(s, d) \in [V]^2$ et $m \in L_{IEML}$. Nous pouvons montrer que l'ensemble de tous les circuits sémantiques O_c forme un *groupoïde* qui retient les symétries décrites à la section 5.3.4.3.

Démonstration. L'opération de permutation binaire $\otimes$ est partielle [12], ce qui implique qu'elle est définie pour quelques-uns des membres de O_c, mais pas pour tous. L'opération $\otimes$ sur l'ensemble O_c a les propriétés suivantes :

- $\forall a, b, c \in O_c,\ (a \otimes b) \otimes c \to a \otimes (b \otimes c)$
- $\forall a \in O_c,\ \exists a^{-1} \in O_c,\ a \otimes a^{-1} = i_a$
- $\forall a, b \in O_c,\ a \otimes b \to a \otimes b \otimes b^{-1} = a$
- $a \otimes a^{-1} \otimes b = b$; $\forall a \in O_c,\ a \otimes a^{-1} \in O_c$

Ces propriétés sont nécessaires et suffisantes pour caractériser un groupoïde. □

Un sous-ensemble particulièrement important des circuits sémantiques est le sous-ensemble qui préserve la structure de Catégorie [13] (voir la sous-section 5.3.4) donnée par l'équation 5.22. Considérons un sous-ensemble de O_c tel que $\forall a \in O^l_c \subset O_c,\ s = m = d = 3^l,\ 0 \leq l \leq 6$. La collection de O^l_c pour

12. Weisstein, Eric W. "Partial Function." From *MathWorld, A Wolfram Web Resource.* http://mathworld.wolfram.com/PartialFunction.html

13. Dans tout ce paragraphe, le mot Catégorie (avec un C majuscule) est pris au sens mathématique classique et *non pas* au sens technique d'ensemble de séquences de même couche qu'il a en topologie sémantique IEML.

$0 \leq l \leq 6$, avec une collection de morphismes $M_c = \{m_c \mid m_c = (o_c^l \rightarrow o_c^{l+1});\ 0 \leq l \leq 5\}$ et la fonction de triplication de l'équation 5.3 appliquée à (s, m, d) forme la Catégorie $\mathbb{S}$.[14] Nous pouvons alors montrer que le foncteur F de la Catégorie $\mathbb{C}$ vers la Catégorie $\mathbb{S}$ est une application (une image) de $\mathbb{C}$ dans $\mathbb{S}$ telle que : $a_c \in O_c :\ F(a_c) \in O_s$; $(x_c \rightarrow y_c) \in M_c :\ (F(x_c) \rightarrow F(y_c)) \in O_s$ où $\forall a \in O_c,\ F(1_a) = 1_{F(a)}$ et $\forall a, b \in M_c,\ F(a \circ b) = F(a) \circ F(b)$. Ce point est fondamental puisqu'il énonce que le langage IEML considéré comme une Catégorie a une image dans la Catégorie des circuits sémantiques.

5.6.4 Rhizomes

Un rhizome $G_\rho(V, E)$ est un type de circuit sémantique dans lequel $V = \bigcup_i V_i$ et $\forall m, n \in V_i,\ \exists (m, n) \in E \wedge \exists (n, m) \in E$. Ce type de circuit sémantique est donc composé de sous-graphes pleinement connectés, ou *cliques*.

5.6.4.1 Rhizome sériel

Un rhizome sériel est un rhizome $G_s = (V_s, E_s)$ pourvu des propriétés suivantes : G_s est un chemin (voir la sous-section 5.5.1.5), $V_s \subseteq L^P_{IEML}$, $\forall v_i, v_j, v_k \in V_s$, $|v_i| = |v_j|$, et il existe une relation binaire $\leq$ sur V_s telle que : $v_i \leq v_j \wedge v_j \leq v_i \rightarrow v_i = v_j$, $v_i \leq v_j \wedge v_j \leq v_k \rightarrow v_i \leq v_k$, $v_i \leq v_j \vee v_j \leq v_i$.

5.6.4.2 Rhizome étymologique

Un rhizome étymologique est un rhizome $G_e = (V_e, E_e)$ pourvu des propriétés suivantes : G_e a une structure d'arbre (voir la sous-section 5.5.1.3), $V_e \subseteq L^P_{IEML}$, $\forall e_i, e_j \in E_e$, $e_i \neq e_j$, $e_i \in$ `substance` $\cup$ `attribute` $\cup$ `mode` tiré des équations 5.6, 5.7 et 5.8.

5.6.4.3 Rhizome taxinomique

Un rhizome taxinomique est un rhizome $G_t = (V_t, E_t)$ pourvu des propriétés suivantes : G_t est un arbre (voir la sous-section 5.5.1.3), $V_t \subseteq L^P_{IEML}$ et $\forall e_i \in E_t,\ e_i = \{(v_i, v_j) \mid \forall j, v_i = \bigcup_j v_j \ \wedge \ \bigcap_j v_j = \emptyset\}$.

5.6.4.4 Paradigmes

Un paradigme $\mathbb{P}$ est le résultat d'une union, intersection ou différence symétrique de rhizomes de type matriciel, taxinomique, étymologique ou sériel (voir la sous-section 5.5.1.4).

5.6.4.5 Dictionnaire

Un dictionnaire $\mathbb{D}$ est un paradigme dont les sommets ont une correspondance bijective avec des descripteurs en langues naturelles.

14. Voir note précédente.

5.7 Critères quantitatifs

5.7.1 Similarité structurale

5.7.1.1 Généralités

Definition 5.7.1. *Une matrice* $\boldsymbol{A}$ *est un tableau bidimensionnel d'objects de la même classe appartenant à un corps*[15] Ψ*:*

$$\boldsymbol{A} = [a_{ij}] = \begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1N} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{iN} \\ \vdots & & \vdots & & \vdots \\ a_{M1} & \cdots & a_{Mi} & \cdots & a_{MN} \end{bmatrix}, \tag{5.42}$$

$$\forall \left\{ i = \overline{1 \dots N}, j = \overline{1 \dots M}; M, N \in \mathbb{Z} \right\} : a_{ij} \in \Psi \therefore \boldsymbol{A} \in \Psi^{N \times M}, \tag{5.43}$$

où M, N *sont appelés dimensions de la matrice.*

Puisque les entrées a_{ij} de la matrice appartiennent au corpsΨ, pour lequel sont définies les opérations d'addition et de multiplication, nous pouvons définir l'addition et la multiplication de matrices de dimensions compatibles comme suit :

Definition 5.7.2. *Pour* $\boldsymbol{A} \in \Psi^{N \times M}$ *et* $\boldsymbol{B} \in \Psi^{N \times M}$*:*

$$\boldsymbol{A} + \boldsymbol{B} = [a_{ij} + b_{ij}]. \tag{5.44}$$

Definition 5.7.3. *Pour* $\boldsymbol{A} \in \Psi^{N \times M}$, $\boldsymbol{B} \in \Psi^{M \times K}$ *et* $\boldsymbol{C} \in \Psi^{N \times K}$, $\boldsymbol{C} = \boldsymbol{AB}$ *signifie que*

$$[c_{ij}] = \left[\sum_{m=1}^{M} a_{im} b_{mj} \right]. \tag{5.45}$$

On peut montrer que la multiplication de matrices ainsi définie satisfait aux propriétés usuelles de la multiplication : l'associativité [WEI 1999]:

$$(\mathbf{AB})\,\mathbf{C} = \mathbf{A}\,(\mathbf{BC}), \tag{5.46}$$

et la distributivité à gauche et à droite :

$$(\mathbf{A} + \mathbf{B})\,\mathbf{C} = \mathbf{AC} + \mathbf{BC} \tag{5.47}$$

$$\mathbf{C}\,(\mathbf{A} + \mathbf{B}) = \mathbf{CA} + \mathbf{CB}. \tag{5.48}$$

15. Un corps est un anneau (voir 5.3.1) équipé des opérations addition, soustraction, multiplication et division, qui satisfait aux axiomes de clôture, associativité, commutativité, identité (élément neutre), symétrie et distributivité [**?**].

Quant à la commutativité, même si $\mathbf{BA}$ existe, la commutativité n'est pas le cas général [STR 2009]:

$$\mathbf{AB} \neq \mathbf{BA}. \tag{5.49}$$

Les concepts de matrice identité et de matrice inverse sont particulièrement utiles pour les calculs sur les matrices.

Definition 5.7.4. *Une matrice identité $\boldsymbol{I}$ est une matrice qui satisfait à la proposition 5.3.2:*

$$\forall \boldsymbol{A} \in \Psi^{N\times M} \exists \boldsymbol{I} \in \Psi^{M\times M} : \boldsymbol{AI} = \boldsymbol{A} \tag{5.50}$$

Lemma 5.7.1. *Si Ψ est équipé d'un élément scalaire nul $\nvdash$:*

$$\forall a \in \Psi \; \exists \nvdash \in \Psi : a + \nvdash = a \wedge \nvdash \cdot a = \nvdash, \tag{5.51}$$

et d'un élément scalaire identique $\nVdash$:

$$\forall a \in \Psi \; \exists \nVdash \in \Psi : \nVdash \cdot a = a, \tag{5.52}$$

la matrice identité $\boldsymbol{I} = [\iota_{ij}] \in \Psi^{M\times M}$ pour une matrice $\boldsymbol{A} \in \Psi^{N\times M}$ peut alors être calculée comme

$$\iota_{ij} = \begin{cases} \nVdash & \Leftrightarrow i = j \\ \nvdash & \Leftrightarrow i \neq j \end{cases} \tag{5.53}$$

Démonstration. En substituant les propriétés (5.51) et (5.52) dans l'équation (5.45) selon la règle (5.53), nous obtenons

$$\mathbf{AI} = \left[\sum_{m=1}^{M} a_{im}\iota_{mj}\right] = [a_{ij}\iota_{jj}] = [a_{ij} \cdot \nVdash] = [a_{ij}] = \mathbf{A} \tag{5.54}$$

□

Il est trivial de montrer que la multiplication par une matrice identité est un de ces cas particuliers où s'applique la commutativité, puisque la multiplication scalaire qui lui est sous-jacente dans Ψ est commutative :

$$\mathbf{AI} = \mathbf{IA} = \mathbf{A}. \tag{5.55}$$

Pour simplifier l'exposé, nous définirons seulement l'inverse d'une matrice carrée $A \in \Psi^{N\times N}$. [16]

Definition 5.7.5. *La matrice $\boldsymbol{A}^{-1} \in \Psi^{N\times N}$ est appelée une inverse de $\boldsymbol{A} \in \Psi^{N\times N}$ si la propriété suivante s'applique :*

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}. \tag{5.56}$$

Si $\boldsymbol{A}^{-1}$ n'existe pas, $\boldsymbol{A}$ est appelée une matrice singulière.

16. Pour un traitement général, le lecteur est référé au pseudo-inverse de Moore-Penrose dans *Matrix Computations* [GOL 1996].

Un autre concept utile en algèbre matricielle est celui de matrice de permutation qui permet d'identifier les matrices dont les différences peuvent être exprimées comme des permutations de leurs colonnes ou de leurs rangées.

Definition 5.7.6. *La matrice* $\boldsymbol{P}$ *est appelée une matrice de* permutation *si elle satisfait aux conditions suivantes :*

1. $\boldsymbol{P}$ *est une matrice carrée binaire :* $\boldsymbol{P} \in \mathbb{B}^{N\times N}$, $\mathbb{B} := \{0, 1\}$.
2. $\boldsymbol{P}$ *a exactement une entrée identité pour chaque rangée et chaque colonne et des entrées nulles ailleurs :* $\forall i, j = \overline{1 \dots N} : \sum_{i=0}^{N} p_{ij} = 1 \wedge \sum_{j=0}^{N} p_{ij} = 1$.

5.7.1.2 Matrice d'adjacence

Les matrices fournissent une représentation commode pour la manipulation et l'étude des graphes. En particulier la *matrice d'adjacence* d'un graphe représente sa connectivité [CHA 1984].

Definition 5.7.7. *Pour un graphe* $G = (V, E)$, $|E| = N$ *sa matrice d'adjacence est une matrice unique* $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, *où* $\mathbb{R}$ *est l'ensemble des nombres réels. Pour un graphe simple* G *la matrice d'adjacence* $\boldsymbol{A}$ *est calculée de la manière suivante :*

$$a_{ij} = \begin{cases} |v_i v_j| & \Leftrightarrow i \neq j \\ \kappa |v_i v_i| & \Leftrightarrow i = j \end{cases} \tag{5.57}$$

où $|v_i v_j|$ *est le nombre d'arêtes du sommet* i *au sommet* j, *et* $\kappa = 1$ *pour les graphes orientés et* $\kappa = 2$ *sinon.*

5.7.1.3 Isomorphisme de graphes

Les graphes qui ont la même structure et qui diffèrent seulement par des détails insignifiants tels que la numérotation des sommets et des arêtes sont étudiés en utilisant la notion d'isomorphisme[DIE 2005].

Definition 5.7.8. *Deux graphes* G_1 *et* G_2 *avec des matrices d'adjacence* $\boldsymbol{A}_1$ *et* $\boldsymbol{A}_2$ *sont appelées* isomorphes *si et seulement si il existe une matrice de permutation* $\boldsymbol{P}$ *telle que*

$$\boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^{-1} = \boldsymbol{A}_2 \tag{5.58}$$

Une telle définition de l'isomorphie mène naturellement à trouver des moyens de vérifier si une transformation de graphe particulière est ou non isomorphe. Si l'opérateur de transformation génère une matrice qui satisfait à la Définition 5.7.6, alors la matrice est une matrice de permutation et la condition (5.58) est satisfaite automatiquement. D'un autre côté, démontrer une proposition négative (c'est-à-dire que deux graphes G_1 et G_2 ne sont pas isomorphes) requiert la preuve qu'il n'existe pas de permutation qui transforme G_1 en G_2. Une solution à ce dilemme sera fournie plus bas à la sous-section 5.7.1.4 (Lemme 5.7.3).

5.7.1.4 Théorie spectrale des graphes

La définition 5.7.8, quoique mathématiquement rigoureuse, n'est pas très pratique. Parmi les nombreuses applications de la théorie spectrale des graphes, on trouve justement des méthodes plus pratiques pour étudier les propriétés des graphes, et notamment leurs isomorphismes. La théorie spectrale des graphes étudie les propriétés des graphes en relation avec les valeurs propres [GOL 1996] des matrices qui décrivent complètement le graphe, telles que la matrice d'adjacence, la matrice de distances, ou la matrice laplacienne ([WEI 1999]).

Definition 5.7.9. *Un scalaire $\lambda \in \mathbb{C}$ est une* valeur propre *d'une matrice carrée $\boldsymbol{A}$ si elle satisfait l'équation suivante :*

$$(\boldsymbol{A} - \lambda \boldsymbol{I})\, \boldsymbol{q} = 0, \tag{5.59}$$

où $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, $\boldsymbol{I} \in \mathbb{R}^{N\times N}$ et $\boldsymbol{q} \in \mathbb{C}^{N\times 1}$. Le vecteur de colonne $\boldsymbol{q}$ associé à cette valeur propre est appelée un vecteur propre.

En termes scalaires, le système (5.59) a N équations pour $N+1$ inconnues, et donc sa solution $(\lambda, \mathbf{q})$ n'est pas nécessairement unique. Il peut avoir jusqu'à N solutions distinctes qui forment les multi-ensembles $\{\lambda_1, \ldots, \lambda_N\}$ et $\{\mathbf{q}_1, \ldots, \mathbf{q}_N\}$.

Definition 5.7.10. *Un multi-ensemble de valeurs propres $\{\lambda_1, \ldots, \lambda_N\} \in \mathbb{C}$ d'une matrice $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ est appelé le* spectre *de $\boldsymbol{A}$.*

Lemma 5.7.2. *Si $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ est symétrique, alors toutes ses valeurs propres sont des nombres réels :*

$$\forall (i,j) = \overline{1 \ldots N} : a_{ij} = a_{ji} \;\Leftrightarrow\; \{\lambda_1, \ldots, \lambda_N\} \in \mathbb{R} \tag{5.60}$$

et elles ont un ensemble orthogonal de vecteurs propres :

$$\forall (i,j) = \overline{1 \ldots N} : \left\langle \boldsymbol{q}^{(i)}, \boldsymbol{q}^{(j)} \right\rangle \neq 0 \Leftrightarrow i = j \tag{5.61}$$

Corollary 5.7.1. *Les graphes non-orientés ont un spectre réel (cela suit immédiatement de (5.57).*

Definition 5.7.11. *Deux graphes G_1 sont G_2 dits* cospectraux *si et seulement si ils ont un spectre identique $\Lambda(G_1) = \Lambda(G_2)$.*

Lemma 5.7.3. *Les graphes qui ne sont pas cospectraux sont nécessairement non-isomorphes:*

$$\Lambda(G_1) \neq \Lambda(G_2) \Rightarrow \nexists \boldsymbol{P} : \boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^{-1} = \boldsymbol{A}_2 \tag{5.62}$$

Les spectres de matrices peuvent être calculés de manière efficace en utilisant l'algorithme QR pour des matrices relativement petites ou l'algorithme Lanczos pour les grandes matrices clairsemées [GOL 1996]. Avec le résultat du Lemme

Algorithm 5.1 Matrice de calcul de distances pour les graphes non-pondérés

input : Matrice d'adjacence $\mathbf{A} \in \mathbb{Z}^{N \times N}$
output : Matrice de distance $\mathbf{D} \in \mathbb{Z}^{N \times N}$

```
begin
141   B ← I_N ∈ Z^{N×N}   D ← ∞_N ∈ Z^{N×N}   for n = 1 to N do
143     C ← BA   for i = 1 to N do
                  for j = 1 to N do
                    if c_ij > 0 and b_ij = 0 then
144                   d_ij ← n
145     B ← C
```

5.7.3 ceci fournit une méthode pratique pour tester l'absence d'isomorphie entre différents graphes.

La théorie spectrale des graphes est aujourd'hui un champ de recherches fort actif en mathématiques, particulièrement pour les graphes non-orientés. Le lecteur (la lectrice) intéressé peut se référer à [CVE 1997] pour une recension exhaustive de ce domaine.

5.7.2 Mesure de distances

5.7.2.1 Le problème du plus court chemin pour les graphes non-pondérés

Dans l'exploration des graphes il est souvent intéressant de calculer la longueur du plus court chemin entre une certaine paire de sommets ou, plus généralement, entre toutes les paires de sommets.

Definition 5.7.12. *Un graphe $G = (V, E)$ est appelé* non-pondéré *si les distances entre toutes les paires de sommets adjacents (i, j) sont constantes $\forall (i, j) = \overline{1 \dots N}$: $d_{ij} = const$. Dans le cas contraire, nous dirons que le graphe est* pondéré.

Sans perte de généralité, il est souvent pratique de supposer que la distance de base dans un graphe non pondéré est de 1.

Pour les graphes non-pondérés, le problème du plus court chemin peut être résolu d'une manière élégante en calculant la série entière $\mathbf{A}^n$ de la matrice d'adjacence. $\mathbf{A}^n$ possède une propriété intéressante puisque ses entrées $a_{ij}^{(n)}$ sont égales au nombre de chemins de longueur n entre les sommets i et j [CHA 1984]. L'algorithme 5.1 calcule la *matrice de distances* $\mathbf{D} \in \mathbb{Z}^{N \times N}$ qui contient les distances d_{ij} entre les sommets i et j.

L'algorithme 5.1 a une complexité de N multiplications de matrices ou $\mathrm{O}(N^3)$. Cet algorithme fonctionne à la fois pour les graphes orientés et non-orientés et calcule l'ensemble complet des chemins les plus courts entre toutes les paires mesuré en termes de pas entre les sommets.

Algorithm 5.2 Matrice de calcul de distances pour les graphes pondérés

input : $\mathbf{P}^{(0)} \in \mathbb{R}^{N\times N}$, $\mathbf{D}^{(0)} \in \mathbb{R}^{N\times N}$ selon (5.63)
output : $\mathbf{D}^{(N)}$, $\mathbf{P}^{(N)}$

begin
  for $i, j = \overline{1 \dots N}$ **do**
    if $i = j$ **then**
      $p_{ij}^{(0)} \leftarrow$ NaN (IEEE valeur non-numérique)
    else
      $p_{ij}^{(0)} \leftarrow i$
  $k \leftarrow 1$ **repeat**
    for *all* $i, j = \overline{1 \dots N}$ **do**
      $d_{ij}^{(k)} \leftarrow \min\left[\left(d_{ij}^{(k-1)}\right), \left(d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)\right]$ **if** $d_{ij}^{(k)} \neq d_{ij}^{(k-1)}$ **then**
        $p_{ij}^{(k)} \leftarrow p_{kj}^{(k-1)}$
      else
        $p_{ij}^{(k)} \leftarrow p_{ij}^{(k-1)}$
    $k \leftarrow k + 1$
  until $k > N$;

5.7.2.2 Généralisation du problème du plus court chemin

Les circuits sémantiques (Section 5.6.3) peuvent être représentés de manière plus précise par des graphes pondérés dans lesquels des poids l_{ij} sont assignés de manière inversement proportionnelle à l'importance de la connexion entre les ULSs. Pour résoudre le problème des plus courts chemins entre toutes les paires dans les graphes pondérés, définissons la matrice fondamentale des distances $\mathbf{D}^{(0)} \in \mathbb{R}^{N\times N}$ qui contient les longueurs de tous les chemins du premier ordre. Dans la définition qui suit, *arc* désigne une fonction qui vérifie si deux sommets sont adjacents.

Definition 5.7.13.

$$\forall (i,j) = \overline{1 \dots N} : d_{i,j}^{(0)} := \begin{cases} 0 & \Leftarrow i = j \\ l_{ij} & \Leftarrow \exists\ arc(i,j) \\ \infty & \Leftarrow \nexists\ arc(i,j) \end{cases} \tag{5.63}$$

L'algorithme itératif (Algorithme 5.2) [ATA 1998] est attribué à Floyd et Warshall. On l'utilise largement dans de nombreux domaines d'application tels que la génétique comparée, l'analyse de circuits et la recherche opérationnelle [LAR 1997].

Une fois son travail achevé, l'algorithme 5.2 fournit les matrices $\mathbf{D}^{(N)}$ et $\mathbf{P}^{(N)}$ avec $d_{ij}^{(N)}$ contenant la longueur des plus courts chemins entre i et j,

tandis que la matrice $\mathbf{P}^{(N)}$ contient l'information permettant de tracer ces plus courts chemins. La complexité de l'algorithme est égale à $N + 1$ itérations de $\mathrm{O}(N^2)$ chacune, $\mathrm{O}(N^3)$ au total.

Remerciements

Le travail présenté ici a été principalement subventionné depuis 2002 par le programme fédéral des chaires de recherche du Canada. J'ai également reçu deux subventions de recherche du Conseil de recherches en sciences humaines du Canada (CRSH). Je voudrai remercier pour leur collaboration Christian Desjardins (directeur de la société *Ictinus*), qui a programmé la base de données du dictionnaire d'IEML, ainsi que Andrew Roczniak, qui m'a aidé à formaliser la théorie mathématique d'IEML ainsi que les algorithmes de transcodage et qui a relu attentivement le manuscrit. Mon épouse Darcia Labrosse m'a soutenu de toutes les manières possibles durant les nombreuses années où j'ai travaillé à la conception d'IEML. Sans elle, le métalangage IEML n'aurait pu voir le jour.

Index

Bibliographie

[ARZ 1999] Arzi-Gonczarowski Zippora, "Perceive this as that – analogies, artificial perception, and category theory". *Annals of Mathematics and Artificial Intelligence*, 26(1-4) :215–252, 1999.

[ATA 1998] Atallah Mikhail J. (sous la direction de), *Algorithms and Theory of Computation Handbook*. CRC Press, Boca Raton, FL, 1998

[BAC 2000] Bacry Henri (préface de Alain Connes), *La symétrie dans tous ses états*. Vuibert, Paris, 2000

[BAR 2002] Barabasi Albert Laszlo, *Linked, the New Science of Networks.* Perseus publishing, Cambridge, Mass, 2002

[BEA 2002] Béal Marie-Pierre, Carton Olivier "Determinization of transducers over finite and infinite words". *Journal of Theoretical Computer Science*, 289(1) :225–251, October 2002.

[BUT 1991] Butler Gregory, "Fundamental algorithms for permutation groups". *Lecture Notes in Computer Science*, 559, 1991.

[CHA 1984] Chartrand Gary, *Introductory Graph Theory.* Dover Publications, New York, NY, 1984

[CHO 1963] Chomsky Noam; Schützenberger, Marcel P. "The algebraic theory of context free languages", in Braffort, P.; Hirschberg, D.: *Computer Programming and Formal Languages.* North Holland, Amsterdam, 1963, 118-161

[CVE 1997] Cvetkovic Dragos, Rowlinson Peter, Simic Slobodan, *Eigenspaces of Graphs.* Cambridge University Press, Cambridge, UK, 1997

[DIE 2005] Diestel Reinhard, *Graph Theory.* Springer-Verlag, Heidelberg, 2005 En ligne : http://diestel-graph-theory.com

[GIL 1962] Gill A. *Introduction to the Theory of Finite-State Machines.* McGraw-Hill Book Company, 1962

[GOL 1996] Golub Gene H. and van Loan Charles F., *Matrix Computations.* 3rd Edition. The John Hopkins University Press, Baltimore, MD 1996

[HEN 1968] Hennie Frederick C., *Finite-State Models for Logical Machines.* John Wiley & Sons, Inc., 1968.

[HOL 1991] Holzmann Gerard J., *Design and Validation of Computer Protocols*. Prentice Hall, 1991

[HOP 2001] Hopcroft John E., Motwani Rajeev and Ullman Jeffrey D., *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2nd edition, 2001.

[KAP 1994] Kaplan Ronald M. and Kay Martin, "Regular models of phonological rule systems". *Computational Linguistics*, 20(3) :331–378, 1994.

[KLE 1956] Kleene Stephen, *Representation of Events in Nerve Nets and Finite Automata*, Princeton University Press, Princeton, N.J., 1956, pages 3–42

[LAR 1997] Larson Richard C. and Odoni Amedeo R. *Urban Operations Research*. Chapter 6.2: "Travel Distances on Networks", Massachusetts Institute of Technology, Cambridge, MA, 1997-99 http://web.mit.edu/urban_or_book/www/book/

[ROC 1997] Roche Emmanuel, "Compact factorization of finite-state transducers and finite-state automata". *Nord. J. Comput.*, 4(2) :187–216, 1997

[STR 2009] Strang Gilbert, *Introduction to Linear Algebra*. 4th Edition, Wellesley-Cambridge Press, Wellesley, MA 2009

[WEI 1999] Weinstein Eric W., *CRC Concise Encyclopedia of Mathematics*. CRC Press, Boca Raton, FL, 1999